



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

WIRELESS SOFTWARE DEFINED NETWORKING EN MININET WIFI

DOBLE GRADO EN INGENIERÍA EN SISTEMAS DE TELECOMUNICACIÓN
Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

Manuela Calvo Barrios

Tutorizado por Eva María Castro Barbero

Curso académico: 2019/2020

Agradecimientos

En primer lugar, me gustaría agradecer a la Universidad Rey Juan Carlos la oportunidad que me ha brindado de poder realizar en ella los estudios universitarios que yo quería. No ha sido fácil, pero estos cinco años han sido un camino constante de aciertos y aprendizaje que no olvidaré. Con este Trabajo Fin de Grado se cierra una etapa muy bonita, y a la vez se abre otra más madura con nuevas metas.

También me gustaría agradecer al personal docente su dedicación y continua preocupación por la formación de sus alumnos.

Principalmente dedico este trabajo a mis padres, hermano y familiares, por haberme acompañado y apoyado durante toda mi vida y, en especial, en mi etapa universitaria sin importar la distancia. Gracias a vosotros he llegado al final de este camino y me he podido convertir en la persona que soy ahora.

No puedo olvidar agradecer a Dani su compañía y aliento, hace dos años y medio que caminamos de la mano y eres mi gran apoyo.

De especial importancia este tiempo han sido las amistades que conocí en la universidad, que han sabido estar a mi lado cuando más lo necesitaba y se han convertido en la familia que se elige. Así como mis amigas de la infancia, que siempre me animaron a cumplir mis sueños. A todos vosotros os agradezco vuestro tiempo y compañía.

A mi tutora, Eva, por haberme permitido realizar este trabajo, introduciéndome en una parte del mundo de las redes de telecomunicación que desconocía y ha resultado ser todo un descubrimiento. Me ha permitido aprender y querer seguir formándome en este campo. Agradezco tu dedicación en estos tiempos especialmente complicados para todos y aprecio verdaderamente la oportunidad que me has dado de poder realizar un trabajo apasionante.

Resumen

Hace una década surgió el concepto de *Software Defined Networking*, también conocido como redes definidas por software (SDN), con el fin de transformar las redes de telecomunicación tradicionales para que respondan rápida y adecuadamente a los cambios que se producen en ellas. Su objetivo es que la red sea completamente dinámica mediante la separación de plano de control y datos. En la red SDN, el plano de control, apoyado en la figura del controlador, se encarga del diseño de las políticas de red, es decir, dirige el comportamiento de la red mediante el análisis constante de sus parámetros; mientras que el plano de datos, representado por los elementos de red con funciones de encaminamiento, es el encargado de ejecutar las acciones sobre el tráfico en función a los flujos almacenados en sus tablas, proporcionados por el plano de control.

Apoyadas en el estándar establecido mediante el protocolo OpenFlow, las redes definidas por software pretenden ser la revolución que impere en las redes de próxima generación (NGN), sobre las redes actuales basadas en sistemas distribuidos. En este marco se encuadra este Trabajo Fin de Grado. El principal objetivo que se persigue con él es el estudio de configuraciones de red inalámbricas basadas en SDN, usando el protocolo OpenFlow para comunicar al controlador con el resto de dispositivos de encaminamiento.

Para el estudio de este tipo de redes se va a emplear la herramienta de emulación de redes inalámbricas Mininet WiFi, con la que se van a analizar diferentes topologías y configuraciones que permitan conocer el funcionamiento de las redes SDN inalámbricas. En concreto, se han establecido dos escenarios de estudio.

El primero de los escenarios consiste en una topología básica de red que consta de dos puntos de acceso enlazados entre ellos mediante una red inalámbrica mesh. A cada uno de estos puntos de acceso, se conecta también de manera inalámbrica, pero en este caso sin ser enlace mesh, una estación base de la red. Por tanto, la red resultante tiene dos puntos de acceso, dos nodos o estaciones base y el controlador. El objetivo perseguido con el análisis de esta topología es el estudio de los flujos que intercambia el controlador con los puntos de acceso.

El segundo escenario configurado trata de una topología de red más compleja con distintas zonas que se interconectan entre ellas, intentando emular la red de una ciudad pequeña. La finalidad perseguida en el análisis de esta red es comprobar que puede existir conectividad real y que el controlador es capaz de manejar las decisiones pertinentes de manera dinámica en una red más compleja.

Índice general

Agradecimientos

Resumen

| | |
|----------------------------------------------------------------|-----------|
| Índice de figuras | 7 |
| Índice de tablas | 9 |
| Lista de acrónimos y abreviaturas | 11 |
| 1. Introducción | 1 |
| 1.1. Contexto y motivación | 1 |
| 1.2. Objetivos | 2 |
| 1.3. Estructura de la memoria | 2 |
| 2. Estado del Arte | 4 |
| 2.1. Wireless Software Defined Networking | 4 |
| 2.1.1. Origen: la problemática de las redes actuales | 4 |
| 2.1.2. SDN: concepto y arquitectura | 5 |
| 2.1.3. Controlador | 6 |
| 2.1.4. Ventajas de SDN | 8 |
| 2.2. OpenFlow | 9 |

| | | |
|-----------|------------------------------------------------------------------------------|-----------|
| 2.2.1. | Switches de la red | 10 |
| 2.2.2. | Tablas de flujos | 10 |
| | Cabecera | 11 |
| | Acciones | 11 |
| | Estadísticas | 13 |
| 2.2.3. | Canal OpenFlow | 14 |
| | Controller-to-Switch | 14 |
| | Asynchronous | 15 |
| | Symmetric | 16 |
| 2.3. | Mininet WiFi | 17 |
| 2.3.1. | Instalación de Mininet WiFi | 18 |
| 2.3.2. | Primeros pasos en Mininet WiFi | 19 |
| | Creación de topologías de red por comandos en Mininet WiFi | 20 |
| | Scripts Python para crear una red en Mininet WiFi | 20 |
| 3. | Configuración y análisis de una red inalámbrica sencilla | 27 |
| 3.1. | Configuración del script meshAP.py | 28 |
| 3.2. | Flujos instalados en los puntos de acceso | 29 |
| 3.3. | Tráfico OpenFlow entre el controlador y los puntos de acceso | 38 |
| 3.4. | Topología completa resultante | 45 |
| 4. | Evaluación de una red personalizada en Mininet WiFi | 47 |
| 4.1. | Introducción a la topología de red diseñada | 47 |
| 4.2. | Establecimiento de la conexión | 48 |
| 4.3. | Conectividad, latencia y capacidad de las comunicaciones de la red | 52 |
| | 4.3.1. Conectividad y latencia | 52 |
| | 4.3.2. Capacidad y ancho de banda | 53 |
| 4.4. | Análisis de las diferentes áreas de la red | 53 |

| | |
|-----------------------------------------|-----------|
| 4.4.1. WAN | 54 |
| 4.4.2. LAN 1 | 55 |
| 4.4.3. LAN 2 | 56 |
| 4.4.4. LAN 3 | 58 |
| 4.5. Resultados obtenidos | 61 |
| 5. Conclusiones y líneas futuras | 63 |
| 5.1. Conclusiones | 63 |
| 5.2. Diagrama de Gantt | 66 |
| 5.3. Líneas futuras | 67 |
| A. Script meshAP.py | 68 |
| B. Script topo.py | 70 |
| Bibliografía | 74 |

Índice de figuras

| | |
|-------------------------------------------------------------------------------------------------------------|----|
| 2.1. Comparación de arquitectura de red tradicional y SDN. [2] (pág. 4) | 6 |
| 2.2. Estructura del controlador. [2] (pág. 5) | 7 |
| 2.3. Comunicación entre el controlador y dispositivo SDN a través de OpenFlow.[2] (pág. 4) | 10 |
| 2.4. Arquitectura de Mininet WiFi. [8] (pág. 11) | 18 |
| 2.5. Topología lineal con 4 <i>hosts</i> | 20 |
| 2.6. Topología sencilla con 4 <i>hosts</i> | 21 |
| 3.1. Posición de los nodos de la red meshAP. | 28 |
| 3.2. Topología de la red meshAP. | 30 |
| 3.3. Consulta de ap1 al controlador cuando ap1 recibe solicitud de ARP. | 40 |
| 3.4. Respuesta del controlador a ap1 cuando ap1 consulta la recepción de una soli- citud de ARP. | 41 |
| 3.5. Consulta de ap2 al controlador cuando ap2 recibe respuesta de ARP. | 42 |
| 3.6. Respuesta de modificación del controlador a ap2. | 42 |
| 3.7. Respuesta del controlador con las acciones a instalar en ap2. | 43 |
| 3.8. Consulta de ap1 al controlador cuando ap1 recibe ICMP <i>echo request</i> | 44 |
| 3.9. Respuesta del controlador con la modificación del flujo en ap2. | 45 |
| 3.10. Respuesta del controlador con las acciones a instalar en ap1. | 45 |
| 3.11. Topología resultante de la red meshAP. | 46 |
| 4.1. Topología de la red personalizada. | 48 |

| | |
|-------------------------------------------------------------------------|----|
| 4.2. Mensajes intercambiados en la configuración de un AP. | 50 |
| 4.3. Características soportadas por ap1. | 50 |
| 4.4. Puertos disponibles en ap1. | 51 |
| 4.5. Caché ARP de sta1. | 55 |
| 4.6. Caché ARP de sta2. | 55 |
| 4.7. Caché ARP de sta3. | 55 |
| 4.8. Estado inicial STP en los puertos de ap4. | 57 |
| 4.9. Estado STP en los puertos de ap4: STP_FORWARD. | 57 |
| 4.10. Estado STP en los puertos de ap6: uno de ellos STP_BLOCK. | 58 |
| 4.11. Paquete de controlador a ap11. | 60 |
| 4.12. Paquete de controlador a ap9. | 61 |
| 4.13. Paquete del controlador a ap10. | 61 |
| 4.14. Paquete OFPT_FLOW_MOD del controlador a ap11. | 62 |
| 5.1. Diagrama de Gantt del Trabajo Fin de Grado. | 67 |

Índice de tablas

| | |
|-----------------------------------------------------------------------------------------------------------------|----|
| 2.1. Campos de OpenFlow. | 12 |
| 2.2. Acciones opcionales de modificación de campos. | 14 |
| 2.3. Parámetros de configuración de red. | 22 |
| 4.1. Puertos TCP utilizados por los APs para comunicarse con el controlador. | 51 |
| 4.2. Latencias en milisegundos (ms) obtenidas tras el test de conectividad de la red. | 52 |
| 4.3. Capacidad de las conexiones entre los nodos de la red en Mbps. | 53 |
| 4.4. Puertos TCP utilizados por los APs para comunicarse con el controlador en el análisis de la LAN 3. | 59 |

Lista de acrónimos y abreviaturas

API Application Programming Interface

AP *Access Point*

ARP Address Resolution Protocol

HTTP Hypertext Transfer Protocol

IDS Intrusion Detection System

IoT Internet of Things

IP Internet Protocol

LAN Local Area Network

NFV Network Function Virtualization

NGN Next Generation Network

OVS Open Virtual Switch

POP3 Post Office Protocol

QoS Quality of Service

SDN Software Defined Networking

SMTP Simple Mail Transfer Protocol

TCP Transmission Control Protocol

TFG Trabajo Fin de Grado

TLS Transport Layer Security

VoIP Voice over IP

WAN Wide Area Network

WSDN Wireless Software Defined Networking

Capítulo 1

Introducción

1.1. Contexto y motivación

Las necesidades de las redes de telecomunicación han cambiado en los últimos años. Con la llegada de los avances tecnológicos y la programación, se ha conseguido optimizar el reparto que hacen los equipos de los recursos físicos que poseen. Además, en los últimos años otras técnicas como la virtualización de recursos y funciones de red se han implantado en las redes de telecomunicación de la mano de *Network Function Virtualization* (NFV) y *Software Defined Networking* (SDN).

Desde hace aproximadamente una década se trabaja en el desarrollo y posterior implantación de las redes definidas por software o redes SDN. Aunque el objetivo es que este tipo de redes llegue a sustituir a las tradicionales, debido a que permite la optimización de los recursos, obtiene una visión generalizada de la red que facilita la toma de decisiones en el plano de control, y disminuye el número de elementos que son necesarios en ella, aún no se ha conseguido instaurar. La razón principal es que la complejidad de la red puede llegar a ser demasiado elevada cuando hay en ella numerosos nodos y equipos que se comunican entre ellos.

Las redes definidas por software permiten mayor agilidad en el despliegue de configuraciones de red, mayor eficiencia y capacidad de adaptación a las necesidades instantáneas de las aplicaciones.

Este entorno no solo se emplea en redes cableadas, sino que también es aplicable a redes inalámbricas para permitir una gestión unificada tanto de la red cableada como inalámbrica.

En este trabajo se desea analizar el comportamiento de una red SDN con dispositivos

inalámbricos para estudiar cómo se realiza la programación de red con este tipo de escenarios. Dado que no se dispone de tanto hardware para la realización de este análisis, se ha decidido emplear la herramienta de emulación de redes Mininet WiFi.

1.2. Objetivos

El objetivo principal de este trabajo es analizar el comportamiento de redes inalámbricas en entornos de red SDN a través del emulador Mininet WiFi. Para cumplir este objetivo principal se han definido los siguientes subobjetivos:

1. Estudio teórico y práctico de las redes definidas por software.
2. Estudio del protocolo OpenFlow, estándar empleados en este tipo de redes para que se comuniquen el controlador y los elementos de la red.
3. Estudio del funcionamiento del emulador Mininet WiFi.
4. Definición de escenarios de red dentro del emulador Mininet WiFi que permitan analizar el comportamiento de las redes inalámbricas basadas en SDN.
5. Análisis del comportamiento de los distintos dispositivos de red configurados a través del protocolo OpenFlow.

1.3. Estructura de la memoria

El repositorio de GitHub que contiene el material del trabajo se encuentra disponible en este enlace <https://github.com/manuelacb/TFG-Mininet-WiFi>.

El presente Trabajo Fin de Grado se estructura en cinco capítulos. El primero de ellos es el capítulo actual, en el que se describen brevemente cuáles son los aspectos que han motivado su realización y los objetivos que se quieren alcanzar con él. Además proporciona una visión global del mismo. En el segundo capítulo se exponen teóricamente los conceptos que es necesario conocer para poder entender el funcionamiento de las configuraciones analizadas en capítulos posteriores. Estos conceptos que se exponen hacen referencia a la definición, características y arquitectura de las redes SDN, del protocolo OpenFlow, y el funcionamiento de Mininet WiFi, el software empleado para el diseño de las configuraciones. En el tercer capítulo se analiza el

funcionamiento de una topología de red sencilla para conocer cómo se produce el intercambio de información desde el plano de control (controlador) hacia el plano de datos (puntos de acceso) y cómo se instalan las reglas en las tablas de flujos de los puntos de acceso. En el Capítulo 4 se examina una red más compleja formada por distintas zonas con topologías empleadas en las redes tradicionales, con el fin de comprobar la escalabilidad del concepto SDN y su compatibilidad en diferentes situaciones. Por último, se cierra el trabajo con un quinto capítulo que recoge las conclusiones obtenidas tras su realización.

Capítulo 2

Estado del Arte

2.1. Wireless Software Defined Networking

2.1.1. Origen: la problemática de las redes actuales

La red tradicional de Internet usa protocolos distribuidos como IP, TCP, ARP, HTTP, SMTP y POP3, etc. Estos protocolos permiten construir una comunicación transparente entre los componentes heterogéneos que forman la red, mediante la interconexión de los diferentes niveles de comunicaciones que implementan estos protocolos. [1]

A pesar de que estos protocolos cumplen con sus funciones, las redes de telecomunicación siguen sufriendo problemas como la saturación provocada por elevado tráfico o caída de interfaces que actualmente solo pueden ser solventados mediante recursos hardware.

Estos sistemas empleados en la red tradicional no fueron diseñados para soportar una escalabilidad, tráfico y movilidad crecientes de la incipiente *Next Generation Network* (NGN). La NGN o red de próxima generación integrará tráfico y equipos, entre los que se encuentran routers, switches, redes de 3G y 4G, y *access points*, que deben adaptarse a los servicios emergentes de IoT, como VoIP, redes de sensores, *Quality of Service* (QoS), almacenamiento y cómputo en la nube, y otras aplicaciones, para proveer a los usuarios una red segura, estable, veloz y altamente disponible. [2] (pág. 1)

Las limitaciones de las redes tienen su origen, principalmente, en que el procesado de paquetes es realizado en hardware muy específico (plano de datos). Los operadores de red se encuentran también limitados a la hora de administrar su red: la red resultante es muy rígida y

ante situaciones cambiantes que surgen en el día a día solo tienen la capacidad de configurar algunos de sus parámetros, encontrándose a merced de la configuración proporcionada por los fabricantes del hardware o por la realizada por el proveedor de servicios. [2] (pág. 2)

Todos estos aspectos indicados dificultan la evolución de los protocolos de red, haciendo aún más complejo alcanzar el objetivo de red dinámica y adaptativa ante cambios de las condiciones, porque la red del operador se encuentra muy limitada por una tecnología/hardware específico o por el propio proveedor de servicios. [2] (pág. 2)

Es aquí donde entra en juego el concepto de *Software Defined Networking* (SDN). No se trata de algo nuevo o revolucionario, sino que es heredero de premisas ya empleadas en otras arquitecturas de red y protocolos, como las redes activas de los años 90, la separación de plano de control y datos, y el protocolo OpenFlow, que se ha convertido en el estándar para la programación de este tipo de redes. [2] (pág. 2)

2.1.2. SDN: concepto y arquitectura

SDN surgió hace una década con la premisa de solventar todos estos problemas a los que se enfrentan las redes tradicionales basadas en sistemas distribuidos. La arquitectura SDN cuenta con la peculiaridad de la separación entre el plano de control, que se encarga de la toma y el envío de las decisiones de gestión de red, y el plano de datos, que es el encargado del envío del tráfico de la red. La red resultante presenta una estructura más centralizada, en la que las decisiones de gestión y coordinación de los elementos que conforman la red se enfocan en mayor medida a alcanzar las condiciones operativas óptimas para la citada red de telecomunicación. Además, también permite la evolución de nuevos protocolos del plano de datos sin la necesidad de reemplazar el hardware de los switches de la red. [3] (pág. 1)

En los últimos años ha surgido un nuevo concepto que tiene como origen SDN, y en el que la red por completo es inalámbrica, es decir, todas las conexiones entre los elementos de red se realizan a través de medios no guiados. Se conoce como *Wireless Software Defined Networking* (WSDN).

De forma simplificada, SDN propone una visión de red global en la que el plano de control y el plano de datos están separados. La centralización del plano de control permite obtener una visión general de la red. Además, SDN cuenta con una serie de interfaces para que ambos planos, de control y de datos, se comuniquen y puedan intercambiar la información pertinente. En la Figura 2.1 se reflejan las diferencias entre la arquitectura de red tradicional y la red SDN. [2] (pág. 3)

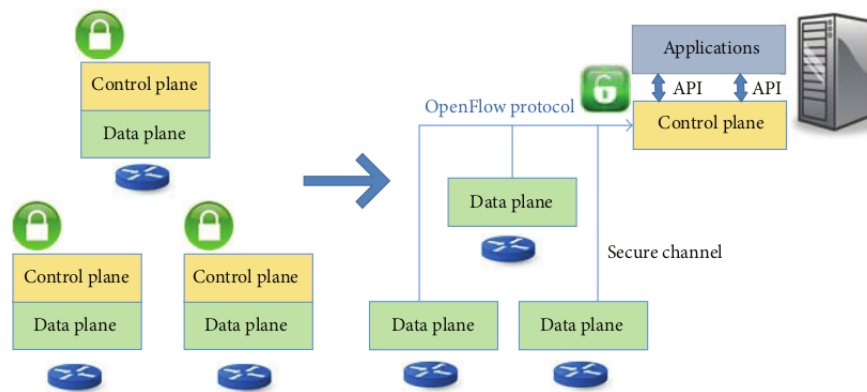


Figura 2.1: Comparación de arquitectura de red tradicional y SDN. [2] (pág. 4)

En la red tradicional basada en sistemas distribuidos, los switches y routers controlan el encaminamiento del tráfico y el reenvío de paquetes. Esta integración vertical del plano de control y datos incrementa la complejidad de la red, dificultando su control y la toma de decisiones. En la red SDN, los switches y routers ejecutan la funcionalidad del plano de datos, pero bajo las órdenes de un nuevo elemento en la red, el controlador. [3] (pág. 1)

Se llama controlador al propio software encargado de la gestión de los recursos disponibles en la red. Los controladores de la red son puntos de gestión que recolectan información de la red para decidir de manera coordinada la configuración de cada uno de sus recursos y elementos. De esta manera se consigue simplificar el papel de los switches y routers, pues únicamente presentan las funciones del plano de datos dentro de la red SDN. Los controladores se ocupan del plano de control como se ha indicado, definen el comportamiento que debe tener la red y responden a los cambios emergentes con decisiones de control, configuración y gestión para encaminar el comportamiento predefinido para ella. [3] (pág. 1)

2.1.3. Controlador

Las redes SDN se diferencian de las redes tradicionales en que existe la figura centralizada del controlador. El controlador se encarga de las funciones del plano de control de la red, es decir, dirige la red con sus decisiones y permite obtener una imagen global de su funcionamiento y su comportamiento. [3] (págs. 2, 3)

El controlador se comunica con otros elementos mediante dos interfaces: la interfaz de bajo nivel *southbound interface* y la interfaz de alto nivel *northbound interface*. Se conoce como

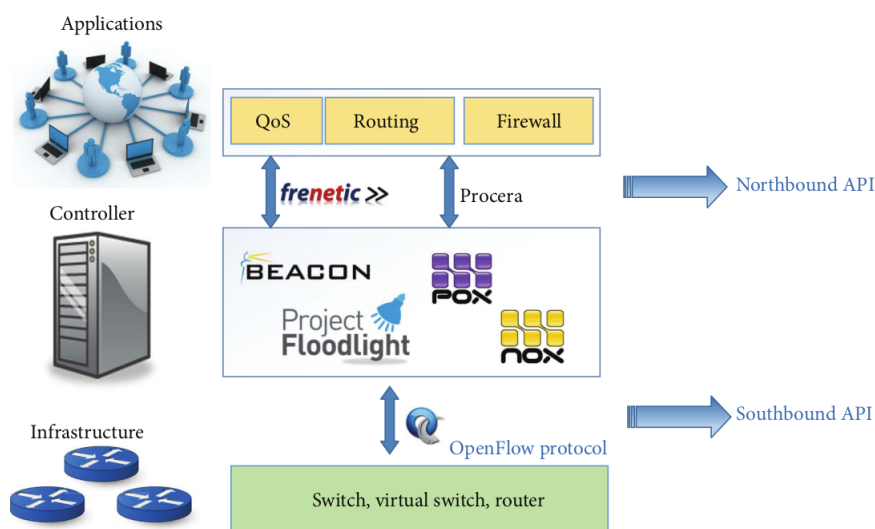


Figura 2.2: Estructura del controlador. [2] (pág. 5)

northbound interface a las aplicaciones de la red, y como *southbound interface* a los servicios de encaminamiento y tratamiento de paquetes de los switches y el hardware de la red. Entre estas dos interfaces se encuentra el software puro del controlador, que se comunica con ellas a través de dos APIs. En la Figura 2.2 muestra cómo es la estructura del controlador y cómo se conecta con las interfaces mencionadas. [2] (pág. 5)

Generalmente, cuando un paquete llega a los switches en los que se encuentran los servicios de reenvío y encaminamiento, el switch consulta la información que tiene almacenada y que establece cómo ha de comportarse su tabla de flujos cada vez que recibe un paquete. Si no tiene en ella ningún flujo que encaje con el paquete recibido, envía un mensaje al controlador para que le indique qué debe hacer con él. El controlador puede tener información que permita dar instrucciones al switch, o puede informar a una aplicación concreta para que decida qué hacer con ese paquete. En esa estructura, el controlador que se encuentra entre las interfaces hace de traductor para permitir la comunicación entre las aplicaciones de red y el dispositivo que encamina el paquete. [4]

Los servicios de nivel bajo pueden ser switches hardware o software. Ambos tipos de switches se comunican con el controlador mediante la interfaz usando OpenFlow. La información que se intercambia por esta interfaz suele corresponder a eventos como la llegada de un paquete que no se sabe encaminar, notificaciones de la red como enlaces caídos, y estadísticas. [4]

El controlador de la red consta de los siguientes servicios: [4]

- Servicio de topología: conoce cómo se conectan los switches (servicios de bajo nivel) de la red y construye una topología de ella.
- Servicio de inventariado: descubre los servicios de la red y guarda información de lo que ofrece cada switch.
- Servicio de estadísticas: almacena los contadores que le reportan los nodos y switches, para formar una idea más aproximada de cómo es la red de forma dinámica.
- *Host tracking*: servicio para descubrir las direcciones IP y MAC de los elementos de la red.

Las aplicaciones que se comunican con el controlador a través del Northbound API ayudan a controlar el comportamiento de la red y a implementar nuevas políticas. Se comunican con el controlador SDN mediante la interfaz de alto nivel, a través de la cuál llega al controlador información del core de la red y órdenes de control del comportamiento de la red. [4]

2.1.4. Ventajas de SDN

Las redes SDN frente a las tradicionales ofrecen una mayor disponibilidad de los servicios ofrecidos. En una red SDN cuando se pierde conectividad con una máquina o conjunto de máquinas, la propia red se encarga de reencaminar el tráfico por otros sistemas e interfaces que se encuentren en funcionamiento, reduciendo así tiempos de desconexión y posibles errores. Por otro lado, la red SDN es más escalable que la tradicional porque es capaz de reconfigurarse dinámicamente según las necesidades de un momento determinado. Otra de las peculiaridades que ofrece SDN es que varios controladores pueden coexistir en una misma red, permitiendo que se establezcan diferentes regiones SDN con sus recursos propios que se comunican entre ellas mediante protocolos *east-west* que se establecen entre los controladores. [4]

En la red tradicional plano de datos y control están unidos. El plano de datos se encarga del manejo de paquete en función de información almacenada en tablas. El plano de control es el que dirige la comunicación entre un nodo y otro mediante protocolos de encaminamiento, como BGP, MPLS o OSPF. En este plano se forman y albergan las políticas de red, se decide cómo se tratan los paquetes y envían esta información al plano de datos. Esto provoca que en la red tradicional sea más difícil intervenir directamente en el plano de datos para modificar el comportamiento de la red, sino que hay que hacerlo sobre las políticas definidas a través de los protocolos de encaminamiento. [4]

Para acceder al plano de control en la red tradicional hay que hacerlo mediante órdenes y comandos de configuración en el hardware específico de encaminamiento, que depende de los fabricantes. Por otro lado, los nodos de la red tradicional son individuales, por lo que cualquier modificación que se quiera hacer en la red deberá hacerse de manera manual sobre los nodos para que surta efecto real, porque no todos los nodos de una red tienen las mismas configuraciones. [4]

Todos estos inconvenientes de las redes tradicionales son ventajas de la arquitectura SDN. Entonces, ¿por qué no se han migrado los servicios de la red tradicional a la nueva red SDN? A pesar de todas las ventajas que presenta el *software defined networking*, su complejidad es muy elevada y no es trivial, ya que posee un alto nivel de abstracción. Además, los equipos de la red tradicional emplean un hardware muy específico que no puede ser usado en SDN. Algunos fabricantes están introduciendo soporte SDN a sus dispositivos de red para poder permitir tanto la configuración tradicional como a través del controlador SDN. [4]

2.2. OpenFlow

Surgió como una alternativa a los protocolos empleados en los estudios de tráfico y redes que se hacían dentro de las universidades, por la facilidad que estas tienen para validar el adecuado funcionamiento de las nuevas tecnologías. OpenFlow se ha convertido en el protocolo de comunicaciones estándar para intercambiar mensajes entre el controlador SDN y un dispositivo de red. [2] (pág. 3)

OpenFlow permite enviar reglas de encaminamiento a los dispositivos de red, estas reglas dotan de comportamiento a estos dispositivos. De este modo, cuando un paquete llega al equipo de red, se lee su cabecera, se consulta la entrada en la tabla de flujos del protocolo OpenFlow y se ejecuta la acción que esté definida para ese tráfico en función de las direcciones origen y destino, el tipo de paquete, su contenido, etc. En la Figura 2.3 se representa la comunicación entre los dispositivos de red y el controlador empleando el protocolo OpenFlow. [2] (pág. 3)

En OpenFlow los switches se conectan de manera lógica con el controlador mediante puertos OpenFlow. Por tanto, solo se pueden enviar y recibir mensajes OpenFlow por los puertos que hayan sido asignados en el switch o máquina a este cometido. [5] (pág. 8)

Los paquetes de datos se reciben en puertos de entrada o recepción (*ingress port*) y se envían por puertos de envío o salida (*output port*). En muchas ocasiones se utilizan los puertos de entrada para consultar la tabla de flujos y decidir qué acción ejecutar sobre el paquete en

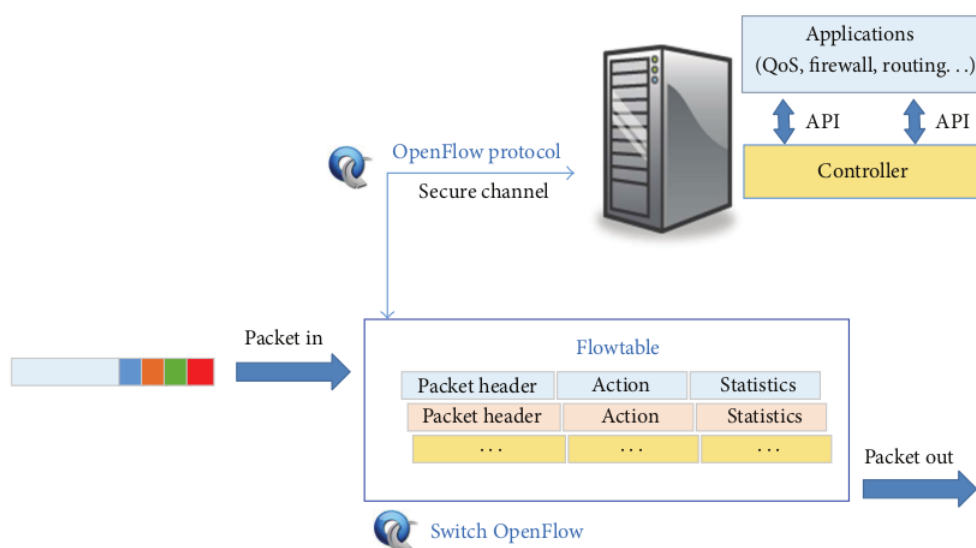


Figura 2.3: Comunicación entre el controlador y dispositivo SDN a través de OpenFlow.[2]
(pág. 4)

función de cuál sea su origen. [5] (pág. 8)

2.2.1. Switches de la red

Los switches de la red que se comunican con el controlador mediante OpenFlow pueden ser de dos tipos, switches puramente OpenFlow, o switches mixtos en los que están activadas las funciones de un switch OpenFlow y además tienen las funciones de routing y switching de las redes tradicionales. [2] (pág. 4)

Los switches híbridos deben tener especificado un mecanismo para clasificar qué tráfico ha de ser tratado por el *pipeline* de OpenFlow y qué paquetes deberán procesarse usando técnicas "tradicionales". Por ejemplo, podría diferenciarse un tráfico de otro mediante el etiquetado de VLANs para la parte de red tradicional y de puerto de entrada para el tráfico de red OpenFlow. [5] (pág. 10)

2.2.2. Tablas de flujos

El formato de OpenFlow está definido para poder establecer reglas de encaminamiento en la llamada tabla de flujos de los dispositivos de red, como por ejemplo son los switches. Esta

tabla tiene principalmente 3 secciones importantes:

- Cabecera del paquete.
- Acción a realizar.
- Algunos datos recopilados y estadísticas para controlar el volumen de tráfico encaminado.

Cabecera

Dependiendo de la versión de OpenFlow soportada en el controlador y en los dispositivos de red, el switch OpenFlow puede procesar un mayor número de campos de paquete. En concreto, en la versión 1.0.0 hay 12 campos, y en la versión 1.3 hay 40 campos. [2] (pág. 4)

Estos campos, que se configuran a través de mensajes OpenFlow, se emplean para consultar las tablas de flujos y encontrar la entrada que coincide con el tráfico recibido, lo que permitirá ejecutar la acción asociada al tipo de paquete. Para que un paquete se considere que cumple con los requisitos de una entrada ha de encajar en todos los campos de ella. No es necesario que los doce campos estén completos en todas las entradas, por lo que puede suceder que un mismo paquete encaje en dos flujos con acciones diferentes. Para evitar estas posibles ambigüedades hay definido un campo de prioridad, en el que se indica cuál es la importancia de esa regla sobre el conjunto de todas las reglas. Estas son solo algunos de los campos incluidos en el protocolo OpenFlow 1.0.0, en la Tabla 2.1 se recogen todos ellos con un breve resumen. [6] (págs. 2, 3, 4)

Acciones

Una vez que el paquete llega al switch OpenFlow y se comparan sus cabeceras para localizar la entrada que le corresponde en la tabla de flujos, se ejecuta la acción indicada para él. Estas acciones pueden ser de dos tipos, principales y opcionales, y marcan el comportamiento del tráfico dentro de la red (plano de datos puro). [2] (pág. 3)

Las acciones principales son las opciones básicas que cualquier switch OpenFlow debe tener configuradas. En concreto, estas acciones indican si un paquete ha de ser enviado por un puerto determinado, si dicho paquete debe ser encapsulado y enviado al controlador para que se instale en el switch un flujo para ese tipo de tráfico, o si el paquete debe ser descartado por el switch. Las acciones opcionales recogen otros comportamientos menos usuales del switch como encolar paquetes a un puerto determinado y otras especificaciones del estándar 802.1D (incluye técnicas de bridging, protocolos y manejo de redes inalámbricas). [2] (pág. 4)

| Campo | Bits | Descripción |
|-------------------------------------------|-------------|------------------------------------------------------------------------------------------------------------------------------|
| Puerto de entrada | - | Número del puerto que ha recibido el paquete |
| Dirección Eth origen | 48 | Máquina origen |
| Dirección Eth destino | 48 | Máquina destino |
| Tipo de paquete Eth | 16 | Es un campo requerido en los switches OpenFlow |
| VLAN | 12 | Para los paquetes Ethernet 0x8100 |
| Prioridad de la VLAN | 3 | Para evitar colisiones entre dos entradas de la tabla |
| Dirección IP origen | 32 | Puede llevar máscara de subred |
| Dirección IP destino | 32 | Puede llevar máscara de subred |
| Protocolo IP | 8 | Protocolo que envía el paquete (TCP, UDP, ICMP, ARP) |
| Bits IP ToS | 6 | Especifica el ToS en todos los paquetes IP |
| Tipo ICMP, puerto de transporte origen | 16 | Indica el puerto de nivel de transporte de la máquina origen que envió el paquete o el tipo del paquete en los mensajes ICMP |
| Código ICMP, puerto de transporte destino | 16 | Código de ICMP o puerto destino del nivel de transporte |

Tabla 2.1: Campos de OpenFlow.

Las entradas de las tablas de flujos están asociadas con acciones que debe ejecutar el switch sobre el paquete recibido. Las que no tienen acciones definidas indican que el tráfico que encaje en ellas deberá ser descartado. También puede haber entradas con varias acciones indicadas, en cuyo caso el switch deberá ejecutarlas siguiendo el orden establecido en ellas. Si alguna de las acciones no puede ser procesada por el switch, esa entrada de la tabla de flujos será ignorada por él, ya que los switches no tienen por qué saber manejar todas las acciones, pero sí que informará al controlador que ha habido un error por flujo no soportado. [6] (pág. 3)

Cuando se arranca un switch OpenFlow dentro de la red, se debe indicar al controlador qué acciones opcionales puede manejar ese switch, y en función de esa información el controlador establecerá las acciones pertinentes para que no haya errores durante el procesamiento de paquetes. [6] (págs. 3, 6)

Las acciones soportadas por los switches son:

- Forward o envío de tráfico: deben soportar el envío de tráfico por puertos físicos y por los siguientes puertos virtuales:
 - ALL: envío a todas las interfaces menos la de entrada.
 - CONTROLLER: envío de paquetes por la interfaz del canal seguro de comunicación con el controlador.

- LOCAL: enviar por la interfaz local del switch.
 - TABLE: poder ejecutar las opciones indicadas en la tabla de flujos.
 - IN_PORT: enviar el paquete por el puerto por el que se recibió en el switch.
- Drop o descarte de tráfico: si no se especifican acciones en una entrada de la tabla de flujos, el tráfico que encaje con ella se descartará.

Por otro lado hay otras acciones consideradas opcionales, que pueden ser soportadas o no en función de cada switch. Son las siguientes:

- Envío de tráfico a través de dos tipos de puertos virtuales:
- NORMAL: empleando las técnicas de envío de las redes tradicionales (VLAN, L2, L3). El switch habitualmente se guiará por el identificador de VLAN para saber qué debe hacer con ese tipo de tráfico según la funcionalidad de la red tradicional.
 - FLOOD: por inundación, se envía el paquete por todos los puertos del switch.
- Encolar un paquete: enviar el paquete a la cola de alguno de los puertos físicos del switch.
- Modificar un campo: permite incrementar el uso de las funcionalidades del protocolo OpenFlow. Las acciones que se pueden implementar son las recogidas en la Tabla 2.2.

Estadísticas

Las estadísticas o contadores pueden recogerse por tabla, por puerto o por cola. Los switches se encargan de recopilarlas y almacenarlas para que el controlador pueda analizar el comportamiento de la red y modificar los parámetros que sean necesarios. Por ejemplo, si al interpretar las estadísticas el controlador detecta una disminución anormal de tráfico procedente de una interfaz, sabrá reconocer que está sucediendo algún problema en esa interfaz o, si no es capaz de identificar con exactitud el origen, en una zona más o menos acotada de la red. Además, también sirven para trazar el comportamiento habitual de la red en función de horas del día, día de la semana o, incluso, estaciones, ya que no suele ser igual el tráfico que hay en la red a las 23:00h que el que hay a las 03:00h, tampoco el de un martes es igual al del sábado, o el de verano al de invierno si se trata de una zona de playa, por ejemplo. Todas estas alternativas tiene que saber reconocerlas el controlador para optimizar el funcionamiento de la red en base a la

| Acción | Bits | Descripción |
|----------------------------|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLAN ID | 12 | Modifica el campo del identificador de VLAN, creándolo si no existiera o actualizándolo al valor indicado. |
| VLAN priority | 3 | Si no existe la VLAN en el paquete, rellena el campo ID con el 0 y añade la prioridad indicada; si la VLAN ya existía, mantiene el ID y modifica su prioridad. |
| VLAN header | | Elimina la cabecera de la VLAN si existiera. |
| Source MAC address | 48 | Cambia la dirección MAC de la máquina Eth origen. |
| Destination MAC address | 48 | Cambia la dirección MAC de la máquina Eth destino. |
| IPv4 source address | 32 | Modifica el valor de la dirección IP origen. |
| IPv4 destination address | 32 | Modifica el valor de la dirección IP destino. |
| IPv4 ToS | 6 | Cambia el valor del ToS. |
| Transport source port | 16 | Actualiza el puerto origen del nivel de transporte (TCP o UDP) y modifica el <i>checksum</i> . |
| Transport destination port | 16 | Actualiza el puerto destino del nivel de transporte (TCP o UDP) y modifica el valor del <i>checksum</i> . |

Tabla 2.2: Acciones opcionales de modificación de campos.

gestión eficiente de los recursos disponibles y del comportamiento que presenta, y las estadísticas sirven para exactamente eso: para optimizar las decisiones tomadas en el plano de control de la red por el controlador. [2] (pág.4)

2.2.3. Canal OpenFlow

Se llama canal seguro de OpenFlow a la interfaz de comunicación entre el controlador y los switches de la red. Los mensajes intercambiados sirven al controlador de la red para configurar las tablas de flujos de los switches, pudiendo de esta manera definir el comportamiento que debe adoptar la red. No solo el controlador envía mensajes, el switch también se comunica con el controlador mediante el canal. Todos estos mensajes del switch y el controlador siguen el formato del protocolo OpenFlow. [6] (pág. 9)

Los paquetes OpenFlow se clasifican en tres tipos: Controller-to-Switch, Asynchronous y Symmetric. A continuación se explican en mayor detalle cada uno de estos tipos.

Controller-to-Switch

Como su nombre indica, es el controlador el que inicia la comunicación con el switch, quien no tiene en todos los casos que responderle. Los mensajes enviados pueden ser: [6] (pág. 10)

- **Características:** se produce cuando en el establecimiento de la sesión entre controlador y switch empleando TLS el controlador envía un mensaje al switch para que le indique sus características. En este caso el switch debe contestar para que el controlador sepa qué acciones soporta.
- **Configuración:** el controlador envía peticiones e instrucciones de configuración al switch. El switch solo debe responder a las peticiones.
- **Modificación de estado:** se utilizan por el controlador para modificar el estado del switch, generalmente mediante la modificación de entradas de la tabla de flujos y establecimiento de las propiedades de los puertos.
- **Lectura de estado:** el controlador las envía para que el switch le reporte las estadísticas que ha recopilado de sus tablas de flujos, puertos y entradas de las tablas de flujos.
- **Envío de paquetes:** son útiles cuando el controlador quiere enviar un paquete por un puerto del switch.
- **De barrera:** se pueden mandar solicitudes y respuestas de este tipo de mensajes, en las que el controlador quiere asegurarse de que los cambios de configuración que debe haber efectuado el switch realmente se han modificado; o para recibir información de las operaciones que han sido completadas.

Asynchronous

Los mensajes asíncronos son aquellos que envía el switch sin petición previa del controlador. El switch suele enviarlos cuando llega un paquete nuevo, cuando cambia su estado o cuando se produce en él un error, por ejemplo, si no puede ejecutar una acción. Principalmente existen cuatro tipos de mensajes asíncronos que envía el switch: [6] (págs. 10, 11)

- **Packet-in:** si llega un paquete al switch que no encaja con ninguna de las entradas de la tabla de flujos, se envía un mensaje al controlador para que instale un flujo nuevo. También se envía el mensaje "packet-in" cuando en la acción definida en la tabla de flujos se indica expresamente que se envíe el mensaje al controlador. En el caso en el que el switch tenga un buffer de almacenamiento de los mensajes enviados al controlador, en ese paquete se envía una parte de la cabecera del paquete recibido y un campo ID indicando cuál es el paquete almacenado en el buffer al que se refiere el mensaje "packet-in". Si el switch no dispone de este buffer, todo el paquete se encapsula dentro del "packet-in".

- **Flow-Removed:** para cada uno de los flujos de la tabla del switch hay dos campos que indican la caducidad del flujo. Uno de ellos especifica la caducidad por desuso del flujo, es decir, tras ese tiempo de inactividad en el flujo, el switch debe eliminarlo. El otro indica cuándo, independientemente de que se haya usado o no, debe ser eliminado un flujo de la tabla. Esto es útil para evitar tablas muy extensas con flujos en desuso, y también para que los flujos estén debidamente actualizados en ella. Por ello, cada vez que el switch elimine alguno, se enviará un mensaje asíncrono "flow-removed" al controlador, para informarle del flujo eliminado. Así mismo, cuando se recibe en el switch una instrucción del controlador para modificar un flujo que ya ha sido eliminado, también el switch envía el mensaje "flow-removed" para que el controlador sepa que está desinstalado.
- **Port-status:** cuando un puerto se cae, es apagado por el usuario o por las especificaciones del estándar 802.1D, entre otros cambios de estado, se envía por parte del switch un mensaje al controlador reportando el cambio en la configuración de la interfaz.
- **Error:** para informar de cualquier error durante el funcionamiento o procesamiento de paquetes en el switch.

Symmetric

No se necesita solicitud o comunicación previa por ninguna de las dos partes para el envío de este tipo de mensajes. Son los siguientes: [6] (pág. 11)

- **Hello:** fase de establecimiento de la conexión entre switch y controlador.
- **Echo:** en forma de solicitud o respuesta, tanto switch como controlador pueden enviar una solicitud y el otro extremo debe responder. Se usan sobre todo para conocer los valores de latencia, ancho de banda disponible y tiempo de vida restante de la conexión establecida.
- **Vendor:** en ellos los switches pueden emplear características adicionales incompatibles con el protocolo OpenFlow desarrollado hasta el momento.

En el capítulo 3 se expondrá el comportamiento de OpenFlow de manera práctica empleando un escenario de red configurado para conocer cómo son las fases del establecimiento de la conexión, qué mensajes realmente se intercambian switch y controlador y cómo funcionan las tablas de flujos de los switches.

2.3. Mininet WiFi

Mininet WiFi es un emulador de redes SDN inalámbricas. Es una evolución del emulador de redes SDN Mininet, que se utiliza en la emulación de redes cableadas. Los emuladores son empleados por los investigadores para probar el comportamiento de una red antes de su despliegue. Tratan de imitar el comportamiento de los sistemas hardware y permiten ejecutar acciones similares a las que se ejecutarían en uno de estos sistemas. Los emuladores ofrecen una interfaz que permite realizar experimentos con la premisa de obtener resultados con un comportamiento lo más realista posible y basado en los fundamentos teóricos predefinidos. En el campo de los experimentos también existen los simuladores y los bancos de pruebas (*testbeds*). Los simuladores son menos realistas que los emuladores porque se basan en conceptos y premisas teóricas de comportamientos para extraer los resultados de los experimentos. Por otro lado, los bancos de pruebas se asemejan en mayor medida a los resultados que obtendría la red una vez desplegada. Sin embargo, los bancos de pruebas cuentan con el inconveniente de que es necesario repetir una operación en reiteradas ocasiones para conocer cuál es el comportamiento promedio para obtener los resultados. [7] (pág. 13)

Mininet WiFi fue creado principalmente para emular redes WiFi del estándar 802.11 de IEEE, pero también se puede usar para emular otros tipos de redes. Permite crear redes virtualizando estaciones, puntos de acceso, hosts, switches y el controlador OpenFlow. Otra de las ventajas de Mininet WiFi es esta, permite crear una red SDN que base su intercambio de paquetes en el protocolo OpenFlow, uno de los más extendidos dentro del estudio de este tipo de redes. Además, emplea la versión 1.0.0 de este protocolo, que es la más extendida y ampliamente utilizada. [7] (pág. 14)

Para su funcionamiento, Mininet WiFi emplea las funcionalidades de virtualización otorgadas por la característica del sistema operativo: *linux namespaces*. Los *linux namespaces* son entornos virtuales que permiten virtualizar una máquina o espacio dentro de una máquina física, para que actúen como si fueran máquinas físicas reales con sus propias rutas, firewalls y servicios de red. Por un lado, se utiliza *linux network namespaces* para virtualizar diferentes interfaces de red existentes en una máquina física, por otro lado también se utilizan para independizar carpetas entre las máquinas virtualizadas. En la Figura 2.4 se muestra la arquitectura de Mininet WiFi sobre los *linux namespaces*. En la figura se muestran tres *network namespaces*: root, host/sta1 y host/sta2, para virtualizar un punto de acceso y dos estaciones base. [7] (pág. 15)

Es conveniente saber que Mininet WiFi tiene los siguientes tipos de componentes de la red:

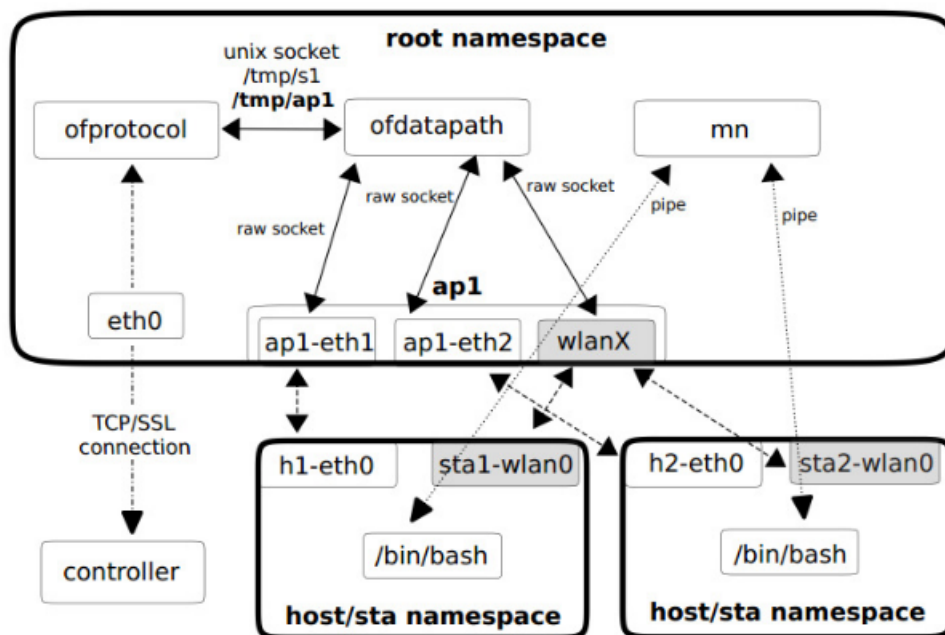


Figura 2.4: Arquitectura de Mininet WiFi. [8] (pág. 11)

- Controlador
- Puntos de acceso
- *Hosts*
- Estaciones base
- Routers

Los *access points* (APs) o puntos de acceso son los switches de la red que se comunican con el controlador y con los *hosts* y estaciones base. Los *hosts* son máquinas finales de la red que pueden tener conexión directa con un AP sin tener que pasar por una estación base previamente. Las estaciones base son elementos de red que dan servicio a los usuarios de la misma y se comunican con el AP para encaminar el tráfico.

2.3.1. Instalación de Mininet WiFi

Mininet WiFi se encuentra disponible para su descarga en el siguiente repositorio de GitHub <https://github.com/intrig-unicamp/mininet-wifi>. También se puede descargar en el

sitio oficial <https://mininet-wifi.github.io/>, donde además se ofrecen algunos tutoriales y explicaciones que facilitan su uso.

Para instalarlo desde GITHUB solo basta con ejecutar los siguientes comandos en la terminal de una máquina Linux:

```
git clone https://github.com/intrig-unicamp/mininet-wifi
cd mininet-wifi
sudo util/install.sh -Wlnfv6
```

Tras estos pasos ya se tendría descargado e instalado el emulador Mininet WiFi en una máquina Linux. Alternativamente, también se podría descargar como se ha indicado anteriormente del sitio oficial de Mininet WiFi. Esta sería una mejor opción si lo que se desea es que el emulador esté dentro de una máquina virtual, pues directamente se ofrece la opción de descargar dicha máquina virtual en la que ya se encuentra instalado el software de Mininet WiFi.

En caso de que se haya preferido la primera opción, para mantener el entorno actualizado a las diferentes versiones que se vayan subiendo al sitio inicial de GitHub bastaría con ejecutar los siguientes comandos en un terminal de la máquina virtual:

```
git pull
sudo make install
```

Tras estos pasos ya estaría disponible el emulador para su uso. A continuación se ofrece un resumen de su funcionamiento y algunos comandos útiles para su uso. En los siguientes capítulos se explica en mayor profundidad cómo funciona el controlador y el envío de mensajes OpenFlow que va configurando el comportamiento de la red que se desea.

2.3.2. Primeros pasos en Mininet WiFi

De las dos formas de instalación del emulador se ha escogido la primera para el desarrollo de este trabajo. Por tanto, todas las configuraciones y comandos empleados funcionan adecuadamente empleando una terminal que se ejecuta directamente sobre Linux. No obstante, en el caso de haber empleado la máquina virtual no debería cambiar el funcionamiento de la herramienta.

Mininet WiFi ofrece diversas configuraciones de red predefinidas en scripts Python ya programados y otras que se pueden crear al vuelo con la ejecución de unos comandos determinados. Además, también ofrece la posibilidad de crear escenarios de movilidad y de establecer el modelo de propagación deseado (para redes inalámbricas en las que la atenuación del medio

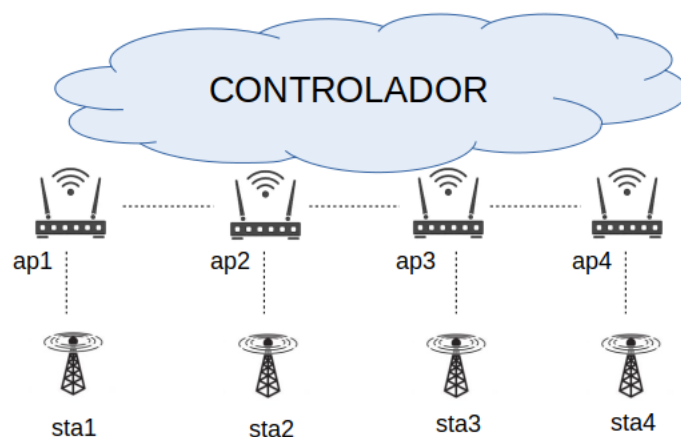


Figura 2.5: Topología lineal con 4 *hosts*.

es importante) en la red. A continuación se detalla en resumen cada una de estas opciones y posibilidades.

Creación de topologías de red por comandos en Mininet WiFi

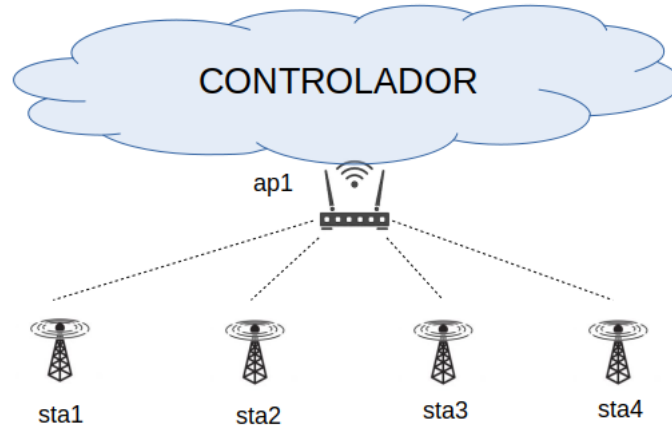
El emulador permite la creación rápida de algunas topologías de red simples bajo la ejecución de determinadas órdenes en la terminal. Se pueden crear topologías lineales, en las que hay un *access point* para cada uno de los *hosts* o estaciones base; y topologías simples en las que un mismo AP da acceso a uno o a varios de los *hosts* o estaciones base de la red.

Para crear la topología de la Figura 2.5 basta con ejecutar el comando `sudo mn -wifi --topo linear,4` en el terminal. De esta manera Mininet WiFi crea un escenario con 4 *hosts*, 4 APs y un controlador. Si por el contrario se quiere configurar una topología simple como la de la Figura 2.6, habrá que ejecutar `sudo mn -wifi -topo single,4` creándose en esta ocasión una red formada por 4 *hosts*, 1 AP y el controlador.

A estos comandos se pueden añadir otros parámetros para configurar la red según los requerimientos del usuario. La Tabla 2.3 recoge todas las opciones de configuración de parámetros y sus valores que permite emplear Mininet WiFi.

Scripts Python para crear una red en Mininet WiFi

Para aquellos desarrolladores o investigadores que prefieran configurar sus propias topologías de red, Mininet WiFi también ofrece esta posibilidad mediante la creación de scripts en

Figura 2.6: Topología sencilla con 4 *hosts*.

| Parámetro | Valores |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -switch=SWITCH | default ivs lxb ovs ovsbr ovsk user[,param=value...] ovs=OVSSwitch default=OVSSwitch ovsk=OVSSwitch lxb=LinuxBridge user=UserSwitch ivs=IVSSwitch ovsbr=OVSBridge |
| -ap=AP | default ivs lxb ovs ovsbr ovsk user[,param=value...] ovs=OVSAP default=OVSAP ovsk=OVSAP lxb=LinuxBridge user=UserAP ivs=IVSSwitch ovsbr=OVSBridge |
| -host=HOST | cfs proc rt[,param=value...] rt=CPULimitedHost'sched': 'rt' proc=Host cfs=CPULimitedHost'sched': 'cfs' |
| -station=STATION | cfs proc rt[,param=value...] rt=CPULimitedStation'sched': 'rt' proc=Station cfs=CPULimitedStation'sched': 'cfs' |
| -controller=CONTROLLER | default none nox ovsc ref remote ryu[,param=value...] ovsc=OVSController none=NullController remote=RemoteController default=DefaultController nox=NOX ryu=Ryu ref=Controller |
| -link=LINK | default ovs tc tcu wmediumd wtc[,param=value...] ovs=OVSLink default=Link wmediumd=wmediumd tcu=TCULink wtc=TCWirelessLink tc=TCLink |

| Parámetro | Valores |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -topo=TOPO | linear minimal reversed single torus tree[,param=value ...] linear=LinearTopo torus=TorusTopo tree=TreeTopo single=SingleSwitchTopo reversed=SingleSwitchReversedTopo minimal=MinimalTopo |
| -no-bridge | prevent low-level bridging of frames between associated stations in the BSS |
| -w, -wifi | activate wifi |
| -d, -docker | for docker environment |
| -container=CONTAINER | docker: container name |
| -ssh-user=SSH_USER | docker: ssh username |
| -plot | plot 2D graph |
| -plot3d | plot 3D graph |
| -channel=CHANNEL | wireless channel configuration |
| -mode=MODE | wireless mode configuration |
| -ssid=SSID | wireless ssid configuration |
| -c, -clean | clean and exit |
| -custom=CUSTOM | read custom classes or params from .py file(s) |
| -driver=DRIVER | wifi driver nl80211 capwap_wtp |
| -test=TEST | none build all iperf pingpair iperfudp pingall |
| -x, -xterms | spawn xterms for each node |
| -i IPBASE, -ipbase=IPBASE | base IP address for hosts |
| -mac | automatically set host MACs |
| -position | automatically set node Positions |
| -arp | set all-pairs ARP entries |
| -v VERBOSITY, -verbosity=VERBOSITY | info warning critical error debug output |
| -innamespace | sw and ctrl in namespace? |
| -listenport=LISTENPORT | base port for passive switch listening |
| -nolistenport | don't use passive listening port |
| -pre=PRE | CLI script to run before tests |
| -post=POST | CLI script to run after tests |
| -pin | pin hosts to CPU cores (requires -host cfs or -host rt) |
| -nat | adds a NAT to the topology that connects Mininet hosts to the physical network. Warning: This may route any traffic on the machine that uses Mininet's IP subnet into the Mininet network. If you need to change Mininet's IP subnet, see the -ipbase option. |
| -version | prints the version and exits |
| -cluster=server1,server2... | run on multiple servers (experimental) |
| -placement=block random | node placement for -cluster (experimental) |

Tabla 2.3: Parámetros de configuración de red.

Python. Para ello, solo se deben importar las funciones que se vayan a necesitar del entorno.

Se va a tomar como ejemplo el siguiente script que crea una red formada por dos estaciones base y un punto de acceso (switch):

```
1 #!/usr/bin/python
2
3 '''
4 This example shows on how to create wireless link between two stas
5 and one AP
6 The wireless mesh network is based on IEEE 802.11
7 '''
8
9 from mininet.log import setLogLevel, info
10 from mn_wifi.link import wmediumd, mesh
11 from mn_wifi.cli import CLI
12 from mn_wifi.net import Mininet_wifi
13 from mn_wifi.wmediumdConnector import interference
14
15
16 def topology():
17     'Create a network.'
18     net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)
19
20     info('*** Creating nodes\n')
21     sta1 = net.addStation('sta1', mac='00:00:00:00:00:11',
22         position='1,1,0')
23     sta2 = net.addStation('sta2', mac='00:00:00:00:00:12',
24         position='31,11,0')
25     ap1 = net.addAccessPoint('ap1', wlans=2, ssid='ssid1',
26         position='10,10,0')
27     c0 = net.addController('c0')
28
29     info('*** Configuring wifi nodes\n')
30     net.configureWifiNodes()
31
32     info('*** Associating Stations\n')
33     net.addLink(sta1, ap1)
34     net.addLink(sta2, ap1)
35
36     info('*** Starting network\n')
37     net.build()
38     c0.start()
```

```
39 ap1.start([c0])
40
41 info('*** Running CLI\n')
42 CLI(net)
43
44 info('*** Stopping network\n')
45 net.stop()
46
47
48 if __name__ == '__main__':
49     setLogLevel('info')
50     topology()
```

Por tanto, si se quiere crear y configurar una red WSDN, en primer lugar hay que definir que el medio va a ser inalámbrico y presentará interferencias. Este último parámetro no es obligatorio, pero añade realismo a la red que se creará tratando de emular este tipo de redes que suelen estar afectadas por problemas de interferencias, desviaciones de campo, dispersiones, fading, etc.

```
net = Mininet_wifi(link=wmediumd , wmediumd_mode=interference)
```

A continuación se deberían añadir tantos APs, estaciones base y *hosts* como se deseen. Las siguientes líneas en el script se encargan de crear esos elementos:

```
sta1 = net.addStation('sta1', mac='00:00:00:00:00:11', position='1,1,0')
ap1 = net.addAccessPoint('ap1', wlans=2, ssid='ssid1,', position='10,10,0')
h1 = net.addHost('h1', mac='00:00:00:00:00:10', position='20,10,0')
```

Nótese que se debe incluir una línea por cada elemento que se vaya a configurar en la red. Para cada uno además hay que definir su nombre, dirección Ethernet y posición en la que se va a emplazar (en los casos de movilidad ha de definirse un origen y destino del movimiento, pudiendo incluso definirse movimiento aleatorio).

Es muy importante en este punto añadir al controlador de la red. Para ello se emplea la siguiente línea de código.

```
c0 = net.addController('c0')
```

A continuación, se ha de escribir la siguiente línea, que es la encargada de dar la orden para que se configuren los nodos y equipos que conforman la red con los parámetros descritos anteriormente.

```
net.configureWifiNodes()
```

Para definir los enlaces y conexiones inalámbricas entre los equipos y sus interfaces se emplea la función *addLink*. Basta con añadir el nombre del AP y el nombre definido para el elemento *host* o estación que se conecta al AP para que pueda comunicarse con el resto de la red. Es de vital importancia añadir los enlaces al script, ya que si no se hace, los nodos no podrán comunicarse entre ellos, porque debido a la definición de este tipo de redes todas las comunicaciones han de pasar por el AP.

```
net.addLink(sta1, ap1)
```

Para establecer comunicación entre dos elementos sin que intermedie un AP se emplea una configuración especial.

Por último, se incluirían estos comandos para iniciar la configuración de la red definida, el controlador y asociar la comunicación entre el controlador y el AP.

```
net.build()
c0.start()
ap1.start([c0])
CLI_wifi(net)
```

El primero de ellos es el encargado de idear la topología de red necesaria para que el segundo comando pueda iniciar al controlador. Una vez que se encuentra el controlador iniciado se iniciarán cada uno de los APs que se hayan definido para la red asociados al controlador. Por último hay que lanzar el cliente de redes inalámbricas para que se virtualicen las tarjetas de red de los elementos. Ahora ya estaría listo el script para su ejecución.

Según las funciones empleadas, al comienzo del script, tras una primera línea `#!/usr/bin/python`, que indica el intérprete a utilizar para la ejecución del script, se deben importar las mismas de cada uno de los módulos a los que pertenecen. En concreto en este caso se han necesitado las siguientes:

```
from mininet.log import setLogLevel, info
from mn_wifi.link import wmediumd, mesh
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi
from mn_wifi.wmediumdConnector import interference
```

Para ejecutarlo habría que escribir en la terminal la siguiente línea que ejecuta el script con los permisos del administrador, necesarios para Mininet WiFi:

```
sudo python nombre_script.py
```

A continuación saldrán varias líneas indicando que la red se está creando y configurando. Una vez se termine de configurar el escenario definido en el script dentro del entorno Mininet, se mostrará un *prompt* dónde se podrán escribir los comandos deseados para ver la configuración de la red. Estos comandos se detallarán durante el desarrollo de los próximos capítulos a la vez que se irá viendo su funcionamiento.

Capítulo 3

Configuración y análisis de una red inalámbrica sencilla

En este capítulo se va a estudiar y analizar el comportamiento de una red mesh sencilla formada por dos estaciones base: sta1 y sta2, y dos puntos de acceso: ap1 y ap2. El objetivo es definir esta topología sencilla a través de un script para Mininet WiFi y estudiar los mensajes de OpenFlow cuando hay un intercambio de datos entre las estaciones de la red. El controlador enviará el comportamiento a los elementos de la red a través de mensajes OpenFlow para permitir la comunicación entre las estaciones.

Dentro de Mininet WiFi no se podían conectar directamente dos puntos de acceso de forma inalámbrica. Para poder realizar esta sencilla configuración fue necesario utilizar la conexión de redes tipo mesh.

Las redes mesh surgieron hace unos cuarenta años para establecer comunicaciones en el ámbito militar, pero posteriormente fueron empleándose en otros entornos como la industria, comunicaciones móviles entre vehículos, redes empresariales o de ciudades y redes de sensores. Su uso se ha ido extendiendo debido a que proporcionan una solución flexible para la comunicación entre múltiples nodos radio, al proporcionar una mayor redundancia y capacidad. [9] (pág. 2)

La tecnología de los nodos mesh se basa en el nivel 2 del modelo OSI, conocido como nivel de datos, y sobre él se produce la encapsulación mesh. Una red mesh se basa en varios elementos conectados entre sí que intercambian continuamente información de gestión para autoconfigurar la red de manera dinámica. [9] (pág. 3)

Para este primer escenario no se requería usar las características propias de las redes mesh, pero fue la única forma de establecer la conectividad entre los puntos de acceso.

3.1. Configuración del script meshAP.py

Una vez definido el concepto de redes mesh se puede proceder a su estudio. El fichero meshAP.py contiene la configuración de una red mesh simple que cuenta con dos APs y dos estaciones base. Dicha configuración corresponde a la de una red SDN en la que los *access points* funcionan como switches típicos de este tipo de redes cuyo comportamiento será gestionado a través de los mensajes OpenFlow que envíe el controlador. El contenido del script meshAP.py se encuentra disponible en su totalidad para su consulta en el Apéndice A.

En la Figura 3.1 se representa la posición de los nodos y APs de la red, obtenida mediante el módulo de representación que ofrece el entorno Mininet WiFi.

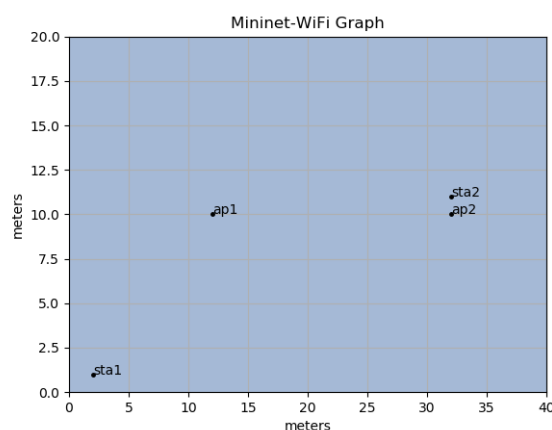


Figura 3.1: Posición de los nodos de la red meshAP.

Se trata de una red mesh de medio inalámbrico (`link=wmedium`) que presenta interferencias (`wmedium mode=interference`). Mininet WiFi se encarga de calcular el nivel de interferencia en base a la distancia existente entre un nodo y sus nodos adyacentes.

Los dos APs de la red mesh configurada en el script presentan cada uno de ellos dos WLANs (`wlans=2`), una de ellas con configuración mesh. Además se indica las direcciones MAC de las dos estaciones base sta1 y sta2.

Respecto a los enlaces entre los diferentes nodos de la red, se observa que se configura un enlace entre sta1 y ap1 y otro entre sta2 y ap2. En ap1 y ap2 además se configuran las interfaces

ap1-wlan2 y ap2-wlan2 como enlaces mesh en el canal 5 de la red.

```
net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)

sta1 = net.addStation('sta1', mac='00:00:00:00:00:11', position='1,1,0')
sta2 = net.addStation('sta2', mac='00:00:00:00:00:12', position='31,11,0')
ap1 = net.addAccessPoint('ap1', wlans=2, ssid='ssid1', position='10,10,0')
ap2 = net.addAccessPoint('ap2', wlans=2, ssid='ssid2', position='30,10,0')
c0 = net.addController('c0')

net.addLink(sta1, ap1)
net.addLink(sta2, ap2)

net.addLink(ap1, intf='ap1-wlan2', cls=mesh, ssid='mesh-ssid', channel=5)
net.addLink(ap2, intf='ap2-wlan2', cls=mesh, ssid='mesh-ssid', channel=5)
```

Con la información que se tiene hasta el momento (la que aporta el script meshAP.py y la posición de los equipos de la red), se puede determinar que la topología de la red mesh es la indicada en la Figura 3.2, a falta de conocer cuáles son las interfaces que se conectan entre los diferentes equipos.

La posición de sta1 es (1, 1, 0) y la de sta2 (31, 11, 0). Cada una de estas estaciones se conecta a su correspondiente punto de acceso: sta1 tiene conexión inalámbrica con ap1 (10, 10, 0), y sta2 con ap2 (30, 10, 0).

Sin embargo, aún falta conocer cuáles son las interfaces de los nodos, cómo se conectan entre ellas y qué papel tiene en estas conexiones el controlador de la red. En los siguientes apartados del presente capítulo se analizará el funcionamiento de la red del ejemplo en ejecución para comprender cómo funciona realmente una red mesh definida por software y poder completar el esquema de la topología de la red analizada.

3.2. Flujos instalados en los puntos de acceso

El comando pingall se encarga de generar un mensaje ICMP echo request entre todas las estaciones base que componen una red. En la red meshAP definida, por tanto, se generarán 2 *pings*: uno desde sta1 hacia sta2 y otro en el sentido contrario.

Los mensajes *ping* funcionan de la siguiente manera: la estación origen envía un paquete

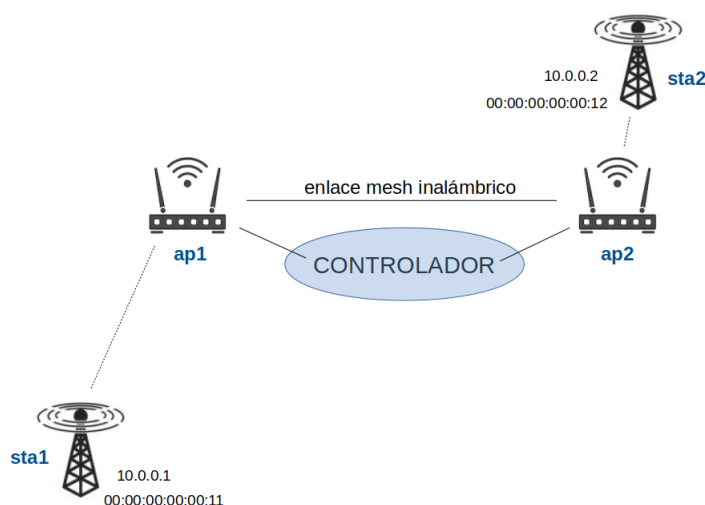


Figura 3.2: Topología de la red meshAP.

ICMP echo request hacia la estación destino, que debe contestar con otro mensaje ICMP echo reply. Si la estación base origen no conoce la dirección MAC de la estación destino, además se generan mensajes ARP request y ARP reply. En el ARP request, la estación origen envía el mensaje a todos los nodos (estaciones base, puntos de acceso, *hosts*) a los que se encuentra conectada, es decir, lo envía a la dirección Ethernet de *broadcast* (ff:ff:ff:ff:ff:ff). En este mensaje, origen pregunta quién tiene la dirección Ethernet asociada a la dirección IP destino del ICMP echo request, y el nodo al que pertenezca dicha IP responde con su dirección MAC en un mensaje ARP reply a la estación origen. Una vez conocida la dirección MAC del destino, el origen envía el ICMP echo request. Cuando el destino lo recibe, puede ocurrir que tampoco tenga la dirección MAC del origen, si se ha borrado de su caché, por lo que tendría que realizar el mismo procedimiento con los mensajes ARP para conocerla y, finalmente, enviar su respuesta ICMP echo reply a esa dirección.

Sin embargo, en las redes SDN el proceso de mensajes se ve un poco alterado. Cuando sta1 envía su ARP request hacia la dirección de *broadcast* para conocer la dirección MAC de sta2, en primer lugar el mensaje llega al *access point* ap1. Seguidamente, ap1 deberá consultar su tabla de flujos para comprobar si existe alguna entrada para tráfico ARP request procedente de sta1 y destinado a sta2. Si tiene alguna entrada que coincida para este tipo de tráfico, sigue las normas o acciones que se indiquen en ella. Si por el contrario, no la tiene, deberá consultar al controlador de la red qué debe hacer en ese caso.

Entre los *access points* y el controlador de una red SDN se intercambian distintos tipos de mensajes. Normalmente el *access point* consulta qué hacer y el controlador le indica a quién debe enviar el tráfico, por qué puerto, si debe eliminarlo, o si tiene que añadir una nueva entrada para ese tipo de tráfico en su tabla de flujos. Estos mensajes entre el controlador y los *access points* se analizan con más detalle en el apartado 3.3 del presente capítulo.

Una vez realizado el intercambio de mensajes ping entre *sta1* y *sta2*, serán analizadas las tablas de flujos que habrá instalado el controlador para marcar el comportamiento de los puntos de acceso.

El *datapath* es la parte del punto de acceso que implementa el plano de datos, es decir, realiza el reenvío de paquetes. En el caso de Mininet WiFi, como todo en realidad se está ejecutando en la misma máquina física, solo existe un *datapath* para los dos APs, que contiene las reglas para el reenvío de tráfico entre todos los puertos del punto de acceso.

El *datapath* tiene los siguientes puertos e interfaces que representan a cada uno de los dispositivos de la red y que se necesitan para analizar los puertos implicados en las reglas del controlador:

```
system@ovs-system:
lookups: hit:79 missed:37 lost:0
flows: 0
masks: hit:153 total:0 hit/pkt:1.32
port 0: ovs-system (internal)
port 1: ap1-wlan1
port 2: ap1-wlan2
port 3: ap1-mp2
port 4: ap1 (internal)
port 5: ap2-wlan1
port 6: ap2-wlan2
port 7: ap2-mp2
port 8: ap2 (internal)
```

El comando anterior muestra que el controlador gestiona los dos puntos de acceso que proporcionan dos redes WiFi 'wlan1' y 'wlan2', y además la red mesh 'mp2'.

Para comprobar los flujos que instala el controlador se debe ejecutar el siguiente comando:

```
mininet-wifi>sh ovs-dpctl dump-flows
```

Flujo 1: indica que todo el tráfico ICMP echo request (icmp_type=8, icmp_code=0) que entre por el puerto 5 del controlador (interfaz ap2-wlan1), con direcciones MAC origen y destino 00:00:00:00:00:12 y 00:00:00:00:00:11, respectivamente, y direcciones IP 10.0.0.2 y 10.0.0.1, origen y destino, salga por el puerto 7, interfaz ap2-mp2.

```
recirc_id(0),in_port(5),eth(src=00:00:00:00:00:12,dst=00:00:00:00:00:11),
eth_type(0x0800),ipv4(src=10.0.0.2,dst=10.0.0.1,proto=1,tos=0/0xfc,frag=no),
icmp(type=8,code=0), packets:0, bytes:0, used:never, actions:7
```

Flujo 2: indica que todo el tráfico ICMP echo reply (icmp_type=0, icmp_code=0) que entre por el puerto 7 del controlador (interfaz ap2-mp2), con direcciones MAC origen y destino 00:00:00:00:00:11 y 00:00:00:00:00:12, respectivamente, y direcciones IP 10.0.0.1 y 10.0.0.2, origen y destino, salga por el puerto 5, interfaz ap2-wlan1.

```
recirc_id(0),in_port(7),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:12),
eth_type(0x0800),ipv4(src=10.0.0.1,dst=10.0.0.2,proto=1,tos=0/0xfc,frag=no),
icmp(type=0,code=0), packets:0, bytes:0, used:never, actions:5
```

Flujo 3: indica que todo el tráfico ICMP echo request (icmp_type=8, icmp_code=0) que entre por el puerto 3 del controlador (interfaz ap1-mp2), con direcciones MAC origen y destino 00:00:00:00:00:12 y 00:00:00:00:00:11, respectivamente, y direcciones IP 10.0.0.2 y 10.0.0.1, origen y destino, salga por el puerto 1, interfaz ap1-wlan1.

```
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:12,dst=00:00:00:00:00:11),
eth_type(0x0800),ipv4(src=10.0.0.2,dst=10.0.0.1,proto=1,tos=0/0xfc,frag=no),
icmp(type=8,code=0), packets:0, bytes:0, used:never, actions:1
```

Flujo 4: indica que todo el tráfico ICMP echo reply (icmp_type=0, icmp_code=0) que entre por el puerto 1 del controlador (interfaz ap1-wlan1), con direcciones MAC origen y destino 00:00:00:00:00:11 y 00:00:00:00:00:12, respectivamente, y direcciones IP 10.0.0.1 y 10.0.0.2, origen y destino, salga por el puerto 3, interfaz ap1-mp2.

```
recirc_id(0),in_port(1),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:12),
eth_type(0x0800),ipv4(src=10.0.0.1,dst=10.0.0.2,proto=1,tos=0/0xfc,frag=no),
icmp(type=0,code=0), packets:0, bytes:0, used:never, actions:3
```

Flujo 5: indica que todo el tráfico ARP request (op=1) que entre por el puerto 3 del controlador (interfaz ap1-mp2), con direcciones MAC origen y destino 00:00:00:00:00:12 y 00:00:00:00:

00:11, respectivamente, y direcciones IP 10.0.0.2 y 10.0.0.1, origen y destino, salga por el puerto 1, interfaz ap1-wlan1.

```
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:12,dst=00:00:00:00:00:11),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.1,op=1/0xff), packets:0,
bytes:0, used:never, actions:1
```

Flujo 6: indica que todo el tráfico ARP request (op=1) que entre por el puerto 5 del controlador (interfaz ap2-wlan1), con direcciones MAC origen y destino 00:00:00:00:00:12 y 00:00:00:00:00:11, respectivamente, y direcciones IP 10.0.0.2 y 10.0.0.1, origen y destino, salga por el puerto 7, interfaz ap2-mp2.

```
recirc_id(0),in_port(5),eth(src=00:00:00:00:00:12,dst=00:00:00:00:00:11),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.1,op=1/0xff), packets:0,
bytes:0, used:never, actions:7
```

Para ver la tabla de flujos del *access point* ap1, hay que ejecutar el siguiente comando:

```
mininet-wifi>sh ovs-ofctl dump-flows ap1
```

Los paquetes ARP reply (arp_op=2) que entran a ap1 por su interfaz ap1-mp2, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ARP 10.0.0.2 y destino ARP 10.0.0.1, saldrán por la interfaz ap1-wlan1 de ap1. No corresponde con ninguna regla del controlador, probablemente ap1 aprenda de la regla instalada para el ARP request.

```
cookie=0x0, duration=5.054s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, arp, in_port='ap1-mp2',vlan_tci=0x0000,d1_src=00:00:00:
00:00:12,d1_dst=00:00:00:00:00:11, arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_
op=2 actions=output:ap1-wlan1
```

Los paquetes ARP request (arp_op=1) que entran a ap1 por su interfaz ap1-mp2, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ARP 10.0.0.2 y destino ARP 10.0.0.1, saldrán por la interfaz ap1-wlan1 de ap1. Corresponde con la regla 5 del controlador.

```
cookie=0x0, duration=9.892s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535,arp, in_port='ap1-mp2',vlan_tci=0x0000,d1_src=00:00:00:
```

```
00:00:12,d1_dst=00:00:00:00:00:11, arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op
=1 actions=output:'ap1-wlan1'
```

Los paquetes ARP reply (arp_op=2) que entran a ap1 por su interfaz ap1-wlan1, cuyo origen es la dirección MAC 00:00:00:00:00:11 y destino 00:00:00:00:00:12, con dirección origen de ARP 10.0.0.1 y destino ARP 10.0.0.2, saldrán por la interfaz ap1-mp2 de ap1. No corresponde con ninguna regla del controlador, probablemente ap1 aprenda de la regla instalada para el ARP request.

```
cookie=0x0, duration=9.889s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, arp,in_port='ap1-wlan1',vlan_tci=0x0000,d1_src=00:00:00
:00:00:11,d1_dst=00:00:00:00:00:12, arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op
=2 actions=output:'ap1-mp2'
```

Los paquetes ICMP echo request (icmp_type=8, icmp_code=0) que entran a ap1 por su interfaz ap1-wlan1, cuyo origen es la dirección MAC 00:00:00:00:00:11 y destino 00:00:00:00:00:12, con dirección origen de ICMP 10.0.0.1 y destino ICMP 10.0.0.2, saldrán por la interfaz ap1-mp2 de ap1. No se corresponde a ninguna regla del controlador.

```
cookie=0x0, duration=5.051s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, icmp,in_port='ap1-wlan1',vlan_tci=0x0000,d1_src=00:00:00:
00:00:11,d1_dst=00:00:00:00:00:12, nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,
icmp_type=8,icmp_code=0 actions=output:'ap1-mp2'
```

Los paquetes ICMP echo reply (icmp_type=0, icmp_code=0) que entran a ap1 por su interfaz ap1-mp2, con dirección MAC origen 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ICMP 10.0.0.2 y destino ICMP 10.0.0.1, saldrán por la interfaz ap1-wlan1 de ap1. No corresponde a ninguna regla del controlador.

```
cookie=0x0, duration=5.036s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, icmp,in_port='ap1-mp2',vlan_tci=0x0000,d1_src=00:00:00
:00:00:12,d1_dst=00:00:00:00:00:11, nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,
icmp_type=0,icmp_code=0 actions=output:'ap1-wlan1'
```

Los paquetes ICMP echo request (icmp_type=8, icmp_code=0) que entran a ap1 por su interfaz ap1-mp2, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ICMP 10.0.0.2 y destino ICMP 10.0.0.1, saldrán por la interfaz ap1-wlan1 de ap1. Corresponde a la regla 3 del controlador.

```
cookie=0x0, duration=5.027s,table=0,n_packets=0,n_bytes=0, idle_timeout=60,
priority=65535, icmp,in_port='ap1-mp2',vlan_tci=0x0000,d1_src=00:00:00
```

```
:00:00:12,dl_dst=00:00:00:00:00:11, nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,
icmp_type=8,icmp_code=0 actions=output:'ap1-wlan1'
```

Los paquetes ICMP echo reply (icmp_type=0, icmp_code=0) que entran a ap1 por su interfaz ap1-wlan1, con origen MAC 00:00:00:00:00:11 y destino 00:00:00:00:00:12, dirección origen de ICMP 10.0.0.1 y destino ICMP 10.0.0.2, saldrán por la interfaz ap1-mp2 de ap1. Corresponde a la regla 4 del controlador.

```
cookie=0x0, duration=5.024s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, icmp, in_port='ap1-wlan1', vlan_tci=0x0000, dl_src=00:00:00:
00:00:11, dl_dst=00:00:00:00:00:12, nw_src=10.0.0.1, nw_dst=10.0.0.2, nw_tos=0,
icmp_type=0, icmp_code=0 actions=output:'ap1-mp2'
```

El mismo comando se ejecuta para ap2:

```
mininet-wifi>sh ovs-ofctl dump-flows ap2
```

Los flujos que se obtienen para ap2 son los siguientes:

Los paquetes ARP reply (arp_op=2) que entran a ap2 por su interfaz ap2-wlan1, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ARP 10.0.0.2 y destino ARP 10.0.0.1, saldrán por la interfaz ap2-mp2 de ap2. No corresponde a ninguna regla instalada en el controlador.

```
cookie=0x0, duration=6.802s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, arp, in_port='ap2-wlan1', vlan_tci=0x0000, dl_src=00:00:00:
00:00:12, dl_dst=00:00:00:00:00:11, arp_spa=10.0.0.2, arp_tpa=10.0.0.1, arp_op=2
actions=output:'ap2-mp2'
```

Los paquetes ARP request (arp_op=1) que entran a ap2 por su interfaz ap2-wlan1, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ARP 10.0.0.2 y destino ARP 10.0.0.1, saldrán por la interfaz ap2-mp2 de ap2. Corresponde a la regla 6 del controlador.

```
cookie=0x0, duration=1.531s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, arp, in_port='ap2-wlan1', vlan_tci=0x0000, dl_src=00:00:00:
00:00:12, dl_dst=00:00:00:00:00:11, arp_spa=10.0.0.2, arp_tpa=10.0.0.1, arp_op=1
actions=output:'ap2-mp2'
```

Los paquetes ARP reply (arp_op=2) que entran a ap2 por su interfaz ap2-mp2, cuyo origen es la dirección MAC 00:00:00:00:00:11 y destino 00:00:00:00:00:12, con dirección origen de ARP 10.0.0.1 y destino ARP 10.0.0.2, saldrán por la interfaz ap2-wlan1 de ap2. No corresponde a ninguna regla del controlador.

```
cookie=0x0, duration=1.520s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535, arp, in_port='ap2-mp2', vlan_tci=0x0000, dl_src=00:00:00:00:00:11, dl_dst=00:00:00:00:00:12, arp_spa=10.0.0.1, arp_tpa=10.0.0.2, arp_op=2 actions=output:'ap2-wlan1'
```

Los paquetes ICMP echo request (icmp_type=8, icmp_code=0, nw_tos=0) que entran a ap2 por su interfaz ap2-mp2, cuyo origen es la dirección MAC 00:00:00:00:00:11 y destino 00:00:00:00:00:12, con dirección origen de ICMP 10.0.0.1 y destino ICMP 10.0.0.2, saldrán por la interfaz ap2-wlan1 de ap2. No corresponde a ninguna regla instalada en el controlador.

```
cookie=0x0, duration=6.788s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535, icmp, in_port='ap2-mp2', vlan_tci=0x0000, dl_src=00:00:00:00:00:11, dl_dst=00:00:00:00:00:12, nw_src=10.0.0.1, nw_dst=10.0.0.2, nw_tos=0, icmp_type=8, icmp_code=0 actions=output:'ap2-wlan1'
```

Los paquetes ICMP echo reply (icmp_type=0, icmp_code=0, nw_tos=0) que entran a ap2 por su interfaz ap2-wlan1, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ICMP 10.0.0.2 y destino ICMP 10.0.0.1, saldrán por la interfaz ap2-mp2 de ap2. No corresponde a ninguna regla instalada en el controlador.

```
cookie=0x0, duration=6.784s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535, icmp, in_port='ap2-wlan1', vlan_tci=0x0000, dl_src=00:00:00:00:00:12, dl_dst=00:00:00:00:00:11, nw_src=10.0.0.2, nw_dst=10.0.0.1, nw_tos=0, icmp_type=0, icmp_code=0 actions=output:'ap2-mp2'
```

Los paquetes ICMP echo request (icmp_type=8, icmp_code=0, nw_tos=0) que entran a ap2 por su interfaz ap2-wlan1, cuyo origen es la dirección MAC 00:00:00:00:00:12 y destino 00:00:00:00:00:11, con dirección origen de ICMP 10.0.0.2 y destino ICMP 10.0.0.1, saldrán por la interfaz ap2-mp2 de ap2. Corresponde con la regla 1 del controlador.

```
cookie=0x0, duration=6.773s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535, icmp, in_port='ap2-wlan1', vlan_tci=0x0000, dl_src=00:00:00:00:00:12, dl_dst=00:00:00:00:00:11, nw_src=10.0.0.2, nw_dst=10.0.0.1, nw_tos=0, icmp_type=8, icmp_code=0 actions=output:'ap2-mp2'
```

Los paquetes ICMP echo reply (icmp_type=0, icmp_code=0, nw_tos=0) que entran a ap2

por su interfaz ap2-mp2, cuyo origen es la dirección MAC 00:00:00:00:00:11 y destino 00:00:00:00:00:12, con dirección origen de ICMP 10.0.0.1 y destino ICMP 10.0.0.2, saldrán por la interfaz ap2-wlan1 de ap2. Corresponde a la regla 2 del controlador.

```
cookie=0x0, duration=6.763s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, icmp,in_port='ap2-mp2',vlan_tci=0x0000,dl_src=00:00:00:
00:00:11,dl_dst=00:00:00:00:00:12, nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,
icmp_type=0,icmp_code=0 actions=output:'ap2-wlan1'
```

Una vez borradas las entradas en las tablas de flujos, se procede a ejecutar nuevamente el comando pingall y se observan algunas diferencias en los flujos instalados.

En el controlador aparecen las siguientes nuevas reglas:

```
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:12,dst=00:00:00:00:00:11),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.1,op=2/0xff), packets:0,
bytes:0, used:never, actions:1
recirc_id(0),in_port(7),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:12),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.2,op=2/0xff), packets:0,
bytes:0, used:never, actions:5
recirc_id(0),in_port(5),eth(src=00:00:00:00:00:12,dst=00:00:00:00:00:11),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.1,op=2/0xff), packets:0,
bytes:0, used:never, actions:7
recirc_id(0),in_port(1),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:12),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.2,op=2/0xff), packets:0,
bytes:0, used:never, actions:3
```

Sin embargo, estos flujos no aparecen inmediatamente tras ejecutar el pingall, sino que hay que esperar varios segundos y volver a cargar las reglas del controlador. Esto se debe a que hay ocasiones en las que la propia estación base, para corroborar que las direcciones MAC que almacena en su caché ARP es correcta, envían ARP request a estas direcciones almacenadas. Por eso aparecen estos flujos unos segundos después de ejecutar el segundo pingall.

En ap1, aparece un flujo relacionado también con este tipo de comunicación de chequeo que hace la estación base:

```
cookie=0x0,duration=6.704s,table=0,n_packets=0,n_bytes=0,idle_timeout
=60, priority=65535,arp,in_port='ap1-wlan1',vlan_tci=0x0000,dl_src=00:00:
00:00:00:11,dl_dst=00:00:00:00:00:12,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,
arp_op=1 actions=output:'ap1-mp2'
```

E igualmente en ap2:

```
cookie=0x0,duration=8.475s,table=0,n_packets=0, n_bytes=0,idle_timeout
=60, priority=65535,arp,in_port='ap2-mp2',vlan_tci=0x0000,dl_src=00:00:
00:00:00:11,dl_dst=00:00:00:00:00:12,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,
arp_op=1 actions=output:'ap2-wlan1'
```

3.3. Tráfico OpenFlow entre el controlador y los puntos de acceso

Para analizar el intercambio de paquetes capturados cuando se ejecuta el comando `pingall` se necesita conocer cómo se identifican los puertos de los puntos de acceso a través de OpenFlow. El comando que nos indica el listado de los puertos de un *access point* es (aplicado para ap1):

```
sh ovs-ofctl show ap1
```

Tras ejecutarlo, en ap1 se obtiene la siguiente salida:

```
OFPT_FEATURES_REPLY (xid=0x2): dpid:1000000000000001 n_tables:254,
  n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS
  ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
  mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(ap1-wlan1): addr:02:00:00:00:02:00
  config: 0
  state: 0
  speed: 0 Mbps now, 0 Mbps max
2(ap1-wlan2): addr:02:00:00:00:03:00
  config: PORT_DOWN
  state: LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max
3(ap1-mp2): addr:02:00:00:00:03:00
  config: 0
```

```

state: 0
speed: 0 Mbps now, 0 Mbps max
LOCAL(ap1): addr:7a:20:a9:fc:24:43
config: PORT_DOWN
state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

```

En ap2:

```

OFPT_FEATURES_REPLY (xid=0x2): dpid:1000000000000002 n_tables:254,
n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS
ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(ap2-wlan1): addr:02:00:00:00:04:00
config: 0
state: 0
speed: 0 Mbps now, 0 Mbps max
2(ap2-wlan2): addr:02:00:00:00:05:00
config: PORT_DOWN
state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
3(ap2-mp2): addr:02:00:00:00:05:00
config: 0
state: 0
speed: 0 Mbps now, 0 Mbps max
LOCAL(ap2): addr:f6:95:f1:7c:34:4a
config: PORT_DOWN
state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

```

Se han realizado capturas en la interfaz de *loopback* para poder almacenar todos los mensajes que intercambian el controlados y los dos puntos de acceso.

El primer ping que se realiza es desde sta1 hacia sta2, por lo que ap1 es el primer punto de acceso que tiene un paquete que no sabe cómo reenviar y por tanto se comunica con el controlador. En un paquete OFPT_PACKET_IN, ap1 consulta qué debe hacer con un paquete procedente la dirección MAC 00:00:00:00:00:11 dirigido a la dirección de *broadcast*, y que entra por el puerto 1 de ap1, correspondiente a la interfaz ap1-wlan1. En la Figura 3.3 se aprecia dicha interacción.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|---------|-------------------|-------------------|----------|--------|-------------------------|
| 559 | 91.0... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 74 | Type: OFPT_ECHO_REPLY |
| 560 | 91.0... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq= |
| 561 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 562 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 563 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq= |
| 564 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 565 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 566 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq= |
| 567 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 568 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 569 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq= |
| 570 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 571 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq= |
| 572 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 573 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |


```

▶ Frame 561: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 48942, Dst Port: 6653, Seq: 6598, Ack: 6344,
▼ OpenFlow 1.0
  .000 0001 = Version: 1.0 (0x01)
  Type: OFPT_PACKET_IN (10)
  Length: 60
  Transaction ID: 0
  Buffer Id: 0xffffffff
  Total length: 42
  In port: 1
  Reason: No matching flow (table-miss flow entry) (0)
  Pad: 00
  ▶ Ethernet II, Src: 00:00:00_00:00:11 (00:00:00:00:00:11), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11)
    Sender IP address: 10.0.0.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.0.2

```

Figura 3.3: Consulta de ap1 al controlador cuando ap1 recibe solicitud de ARP.

En su respuesta, el controlador indica mediante un paquete OFPT_PACKET_OUT que ap1 debe enviar ese tipo de tráfico por el puerto 65531, comportándose como un switch normal al que le llega un paquete dirigido a la dirección de *broadcast*, es decir, copiando ese paquete por el resto de sus puertos. En la Figura 3.4 se muestra el paquete capturado.

Cuando el paquete llega a ap2 a través de ap1, ocurre lo mismo; ap2 envía una consulta al controlador mediante un mensaje OFPT_PACKET_IN para saber qué hacer con el paquete enviado por sta1 a la dirección de *broadcast*, que ha recibido por su interfaz ap2-mp2. El controlador contesta con un paquete OFPT_PACKET_OUT que debe reenviar el paquete por el resto de sus puertos.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-------|---------|-------------------|--------------------|----------|--------|----------------------------------|
| 559 | 91.0... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 74 | Type: OFPT_ECHO_REPLY |
| 560 | 91.0... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6598 Ack= |
| ✓ 561 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 562 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 563 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6658 Ack= |
| 564 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 565 | 93.3... | 00:00:00_00:00:11 | Broadcast | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 566 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6478 Ack= |
| 567 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00:... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 568 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 569 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack= |
| 570 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00:... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 571 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack= |
| 572 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00:... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 573 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |


```

▶ Frame 562: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 48942, Seq: 6344, Ack: 6658, Len: 66
▼ OpenFlow 1.0
  .000 0001 = Version: 1.0 (0x01)
  Type: OFPT_PACKET_OUT (13)
  Length: 66
  Transaction ID: 0
  Buffer Id: 0xffffffff
  In port: 1
  Actions length: 8
  Actions type: Output to switch port (0)
  Action length: 8
  Output port: 65531
  Max length: 0
▶ Ethernet II, Src: 00:00:00_00:00:11 (00:00:00:00:00:11), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11)
  Sender IP address: 10.0.0.1
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 10.0.0.2

```

Figura 3.4: Respuesta del controlador a ap1 cuando ap1 consulta la recepción de una solicitud de ARP.

Hay que destacar que en estos casos el controlador no indica a ap1 ni a ap2 que instalen entradas nuevas para este tipo de tráfico en sus tablas de flujos, sino que únicamente les indica qué debe hacer con ellos. Si se volviera a enviar un paquete desde sta1 dirigido a la dirección de *broadcast*, los *access points* tendrían que volver a consultar al controlador.

Sin embargo, en la respuesta de sta2 al ARP request procedente desde sta1 ocurre algo diferente. sta2 responde con un ARP reply, que llega a ap2, y este realiza la pertinente consulta al controlador mostrada en la Figura 3.5.

En este caso el controlador envía dos paquetes como respuesta a la consulta de ap2. Se trata de un mensaje OFPT_FLOW_MOD, en el que indica a ap2 que debe añadir una nueva entrada a su tabla de flujos para el tráfico que se indica en el mensaje OFPT_PACKET_OUT. Las Figuras 3.6 y 3.7 muestran estos dos paquetes capturados.

Por tanto, ap2 añade un nuevo flujo en su tabla para el tráfico ARP reply procedente de sta2

| | | | | | | |
|-----|---------|-------------------|-------------------|----------|-----|--------------------------------------|
| 567 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 568 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 569 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack=6368 |
| 570 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 571 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack=6434 |
| 572 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 573 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |

▶ Frame 567: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ Transmission Control Protocol, Src Port: 48944, Dst Port: 6653, Seq: 6478, Ack: 6288, Len: 60
 ▼ OpenFlow 1.0
 .000 0001 = Version: 1.0 (0x01)
 Type: OFPT_PACKET_IN (10)
 Length: 60
 Transaction ID: 0
 Buffer Id: 0xffffffff
 Total length: 42
 In port: 1
 Reason: No matching flow (table-miss flow entry) (0)
 Pad: 00
 ▶ Ethernet II, Src: 00:00:00_00:00:12 (00:00:00:00:00:12), Dst: 00:00:00_00:00:11 (00:00:00:00:00:11)
 ▼ Address Resolution Protocol (reply)
 Hardware type: Ethernet (1)
 Protocol type: IPv4 (0x0800)
 Hardware size: 6
 Protocol size: 4
 Opcode: reply (2)
 Sender MAC address: 00:00:00_00:00:12 (00:00:00:00:00:12)
 Sender IP address: 10.0.0.2
 Target MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11)
 Target IP address: 10.0.0.1

Figura 3.5: Consulta de ap2 al controlador cuando ap2 recibe respuesta de ARP.

| | | | | | | |
|-----|---------|-------------------|-------------------|----------|-----|--------------------------------------|
| 567 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 568 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 569 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack=6368 |
| 570 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 571 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack=6434 |
| 572 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 573 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 574 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6538 Ack=6434 |
| 575 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 576 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6538 Ack=6434 |
| 577 | 93.3... | 10.0.0.1 | 10.0.0.2 | OpenFlow | 182 | Type: OFPT_PACKET_IN |
| 578 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 579 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6538 Ack=6434 |

▶ Frame 568: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 48944, Seq: 6288, Ack: 6538, Len: 60
 ▼ OpenFlow 1.0
 .000 0001 = Version: 1.0 (0x01)
 Type: OFPT_FLOW_MOD (14)
 Length: 80
 Transaction ID: 0
 Wildcards: 0
 In port: 1
 Ethernet source address: 00:00:00_00:00:12 (00:00:00:00:00:12)
 Ethernet destination address: 00:00:00_00:00:11 (00:00:00:00:00:11)
 Input VLAN id: 65535
 Input VLAN priority: 0
 Pad: 00
 DL type: 2054
 IP ToS: 0
 IP protocol: 2
 Pad: 0000
 Source Address: 10.0.0.2
 Destination Address: 10.0.0.1

Figura 3.6: Respuesta de modificación del controlador a ap2.

y destinado a sta1, que entra por el puerto 1 de ap2, correspondiente a la interfaz ap2-wlan1, para que sea enviado por el puerto 3 de ap2, interfaz ap2-mp2. En ap1 ocurre lo mismo: tras

| | | | | | | |
|-----|---------|-------------------|-------------------|----------|-----|--------------------------------------|
| 570 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 571 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48944 → 6653 [ACK] Seq=6538 Ack=6434 |
| 572 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 126 | Type: OFPT_PACKET_IN |
| 573 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 574 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6718 Ack=6490 |
| 575 | 93.3... | 00:00:00_00:00:12 | 00:00:00_00:00... | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 576 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6718 Ack=6556 |
| 577 | 93.3... | 10.0.0.1 | 10.0.0.2 | OpenFlow | 182 | Type: OFPT_PACKET_IN |
| 578 | 93.3... | 127.0.0.1 | 127.0.0.1 | OpenFlow | 146 | Type: OFPT_FLOW_MOD |
| 579 | 93.3... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 48942 → 6653 [ACK] Seq=6834 Ack=6636 |


```

▶ Frame 570: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 48944, Seq: 6368, Ack: 6538, Len: 66
▼ OpenFlow 1.0
  .000 0001 = Version: 1.0 (0x01)
  Type: OFPT_PACKET_OUT (13)
  Length: 66
  Transaction ID: 0
  Buffer Id: 0xffffffff
  In port: 1
  Actions length: 8
  Actions type: Output to switch port (0)
  Action length: 8
  Output port: 3
  Max length: 0
  ▶ Ethernet II, Src: 00:00:00_00:00:12 (00:00:00:00:00:12), Dst: 00:00:00_00:00:11 (00:00:00:00:00:11)
  ▼ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 00:00:00_00:00:12 (00:00:00:00:00:12)
    Sender IP address: 10.0.0.2
    Target MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11)
    Target IP address: 10.0.0.1

```

Figura 3.7: Respuesta del controlador con las acciones a instalar en ap2.

su consulta al controlador para ese tipo de tráfico, este le responde con dos mensajes para que instale un nuevo flujo en su tabla.

Se puede comprobar que realmente estos flujos se instalan en los *access points*: mirando el listado de flujos del apartado 3.2 se deben encontrar estas reglas.

En ap1:

```

cookie=0x0, duration=5.054s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, arp, in_port='ap1-mp2',vlan_tci=0x0000,dl_src=00:00:00:
00:00:12,dl_dst=00:00:00:00:00:11, arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_
op=2 actions=output:'ap1-wlan1'

```

En ap2:

```

cookie=0x0, duration=6.802s, table=0, n_packets=0, n_bytes=0, idle_timeout
=60, priority=65535, arp,in_port='ap2-wlan1',vlan_tci=0x0000,dl_src=00:00:00:
00:00:12,dl_dst=00:00:00:00:00:11, arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2
actions=output:'ap2-mp2'

```

De igual manera, cuando sta1 envía el ICMP echo request hacia sta2, al llegar el paquete

a la interfaz ap1-wlan1, ap1 consultará al controlador, como muestra la Figura 3.8.

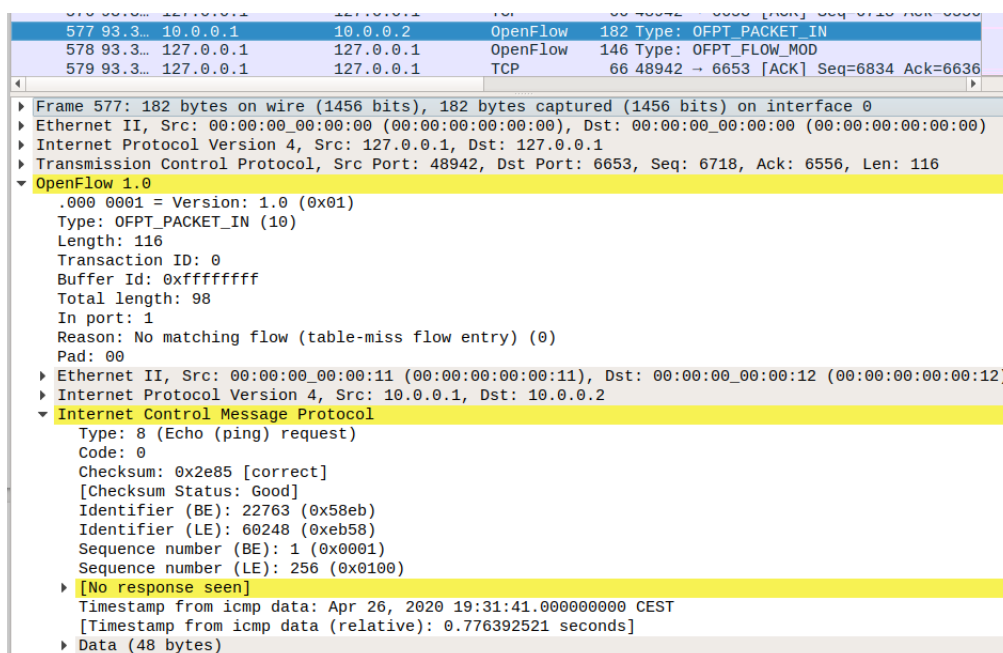


Figura 3.8: Consulta de ap1 al controlador cuando ap1 recibe ICMP echo request.

Igual que en el caso anterior, el controlador contesta a ap1 indicándole que debe instalar un nuevo flujo para el tráfico ICMP echo request procedente de sta1 hacia sta2, que entra por el puerto 1, interfaz ap1-wlan1 y que debe ser enviado por el puerto 3 ap1-mp2. En las Figuras 3.9 y 3.10 se muestran estos mensajes.

Los restantes flujos instalados en los *access points* se añaden a las tablas de flujos de igual manera que los dos últimos casos analizados.

No obstante, a parte de indicar cómo es la instalación de estos flujos, las capturas también aportan información acerca de las interfaces que se conectan entre ellas:

- En sta1 sta1-wlan0 se conecta con ap1-wlan1.
- En ap1, a parte de la conexión indicada anteriormente, ap1-mp2 se conecta con ap2-mp2.
- En ap2, además de la conexión con ap1, ap2-wlan1 se conecta con sta2-wlan0.

Con esta información de las conexiones entre los nodos es sencillo completar el esquema de la topología de la red mostrada en el apartado 3.1. La topología resultante se muestra en el apartado 3.4.

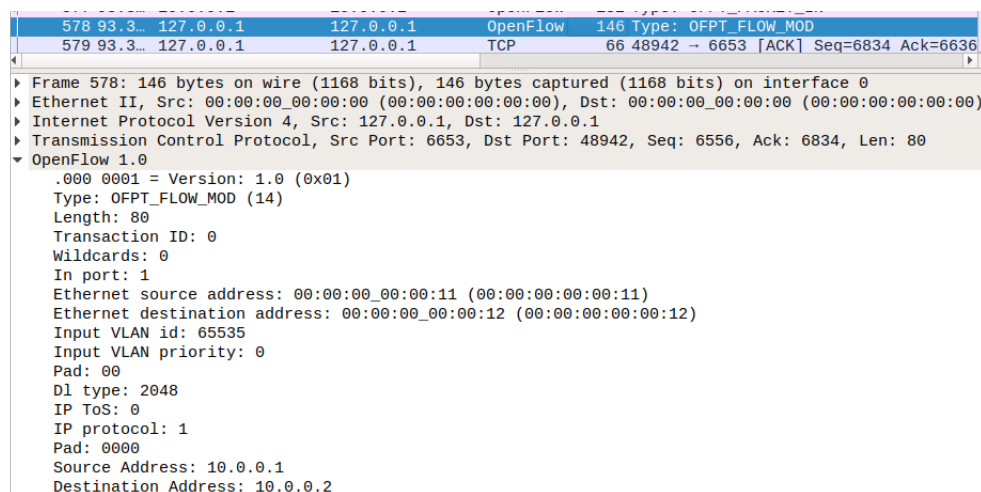


Figura 3.9: Respuesta del controlador con la modificación del flujo en ap2.

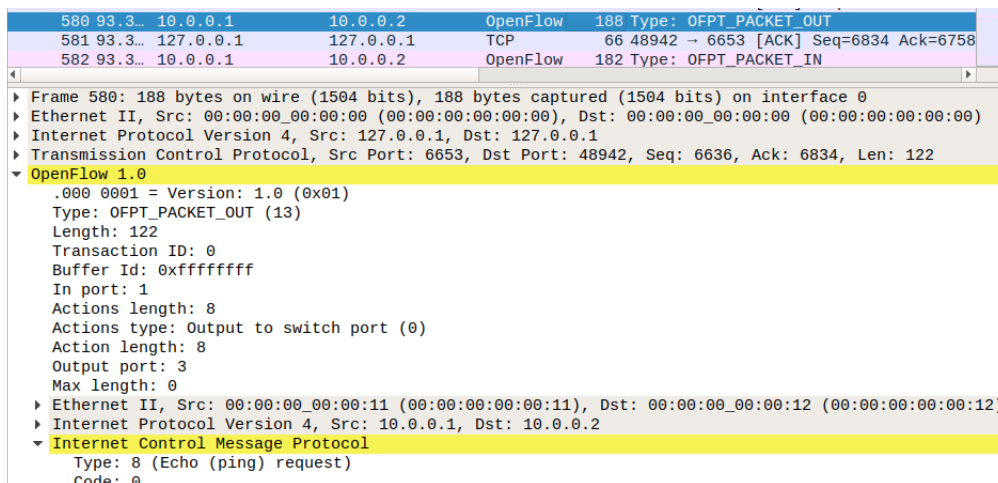


Figura 3.10: Respuesta del controlador con las acciones a instalar en ap1.

3.4. Topología completa resultante

Tras conocer las conexiones de los nodos de la red, se puede completar la topología de la red, mostrada en la Figura 3.11.

Además, se ha añadido la figura del controlador en la representación. Como se ha visto en el anterior apartado 3.3, el controlador está conectado mediante la interfaz de *loopback* con los dos *access points* ap1 y ap2, y por ello, se representa en forma de nube. Realmente se podría considerar a ap1 y a ap2 como un único punto de acceso de la red, dado que están conectados entre ellos mediante las interfaces ap1-mp2 y ap2-mp2, y solo existen en esta red configurada dos

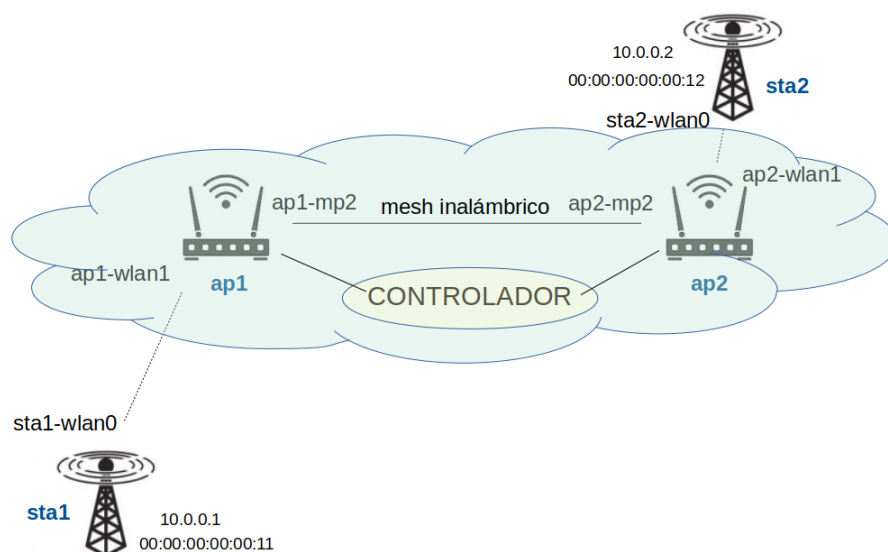


Figura 3.11: Topología resultante de la red meshAP.

estaciones base que se comunican entre ellas. El controlador se comunica con todos los puntos de acceso o switches de la red interconectándolos. Esto es muy útil para, por ejemplo, poder determinar qué nodos de la red están congestionados y crear rutas alternativas que disminuyan la densidad de tráfico en dichos equipos y la red se autoequilibra. Esta es una de las ventajas de este tipo de redes definidas por software, su capacidad de autogestión y autoconocimiento.

Esta red analizada es sencilla y no tiene mucho sentido entonces que haya dos puntos de acceso diferentes. Sin embargo, cuando el tamaño de la red aumenta y hay varias estaciones base que se comunican entre ellas, sí que estaría suficientemente motivado el uso de varios puntos de acceso para, como se ha indicado en el párrafo anterior, poder verificar que realmente la red es capaz de autogestionarse.

No obstante, dentro de la simplicidad que presenta meshAP.py se encuentra la ventaja de haber permitido conocer con mayor grado de detalle cómo funciona la red y qué tareas desempeña el controlador. Si se hubiera intentado analizar una red de mayor envergadura, habría sido más complicado detenerse y fijarse en los detalles de funcionamiento que se han descrito, como las tablas de flujos, que crecerían rápidamente para conectar los nodos de una red compleja, o el intercambio de paquetes OpenFlow entre puntos de acceso y controlador.

Capítulo 4

Evaluación de una red personalizada en Mininet WiFi

4.1. Introducción a la topología de red diseñada

El objetivo de este capítulo es examinar un escenario complejo de una red mesh dentro de Mininet WiFi con distintas áreas dentro de una misma red y protocolos empleados en las redes tradicionales, STP en este caso.

La topología de la red diseñada es la que muestra la Figura 4.1, y trata de representar tres tipos de esquemas empleados en las redes reales actuales: una primera topología en forma de estrella con 3 nodos, un anillo de 4 estaciones y una tercera topología de árbol con 4 nodos. Cabe destacar que cada una de estas topologías mencionadas dentro de la red se correspondería con una red de área local (LAN) y los diferentes puntos de acceso o APs que las conectan constituyen la red de área amplia (WAN).

Para lograr la configuración establecida en la Figura 4.1 han sido necesarias 11 estaciones base en la red, 10 APs, 15 enlaces inalámbricos y 8 enlaces cableados.

De la topología configurada y mostrada en la Figura 4.1 las 8 conexiones cableadas de la red son las que unen entre sí los APs de cada uno de los nodos del bucle de LAN 2, el que une ap4 con ap2, ap3 con ap9, ap9 con ap10 y ap11 con ap9. El resto de conexiones establecidas en la red se realizan mediante radioenlaces que se han emulado empleando el modelo de pérdidas de propagación log-distancia, siendo 5 el exponente de propagación. Con estas características la intención es emular un entorno urbano afectado por *shadowing* (hay obstáculos que impiden

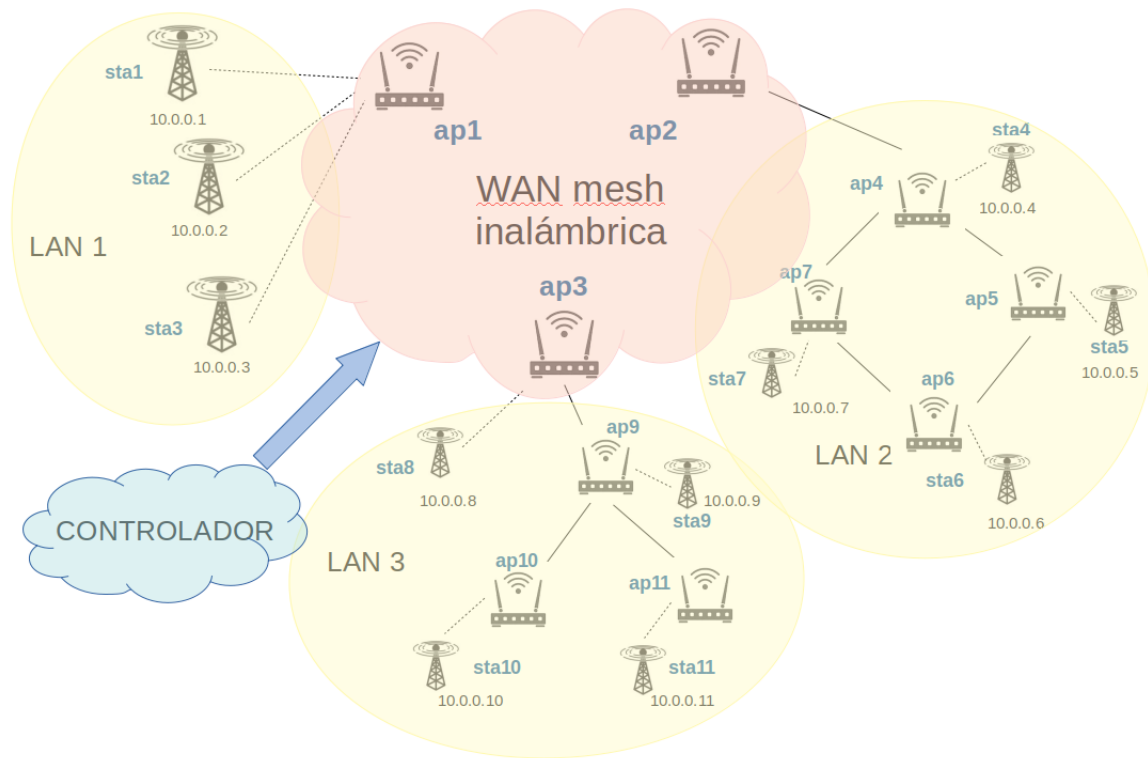


Figura 4.1: Topología de la red personalizada.

que el receptor reciba toda la amplitud de la señal enviada por el equipo transmisor).

4.2. Establecimiento de la conexión

La configuración realizada se encuentra en el Apéndice B. Para el ejecutar el script de configuración basta con ejecutar los siguientes dos comandos:

```
sudo mn -c
sudo python topo.py
```

El primero de ellos borra la configuración residual que puede quedar de anteriores pruebas de scripts y configuraciones. El segundo es el que crea y configura la red según lo establecido en el código Python del fichero topo.py.

Para conocer los diferentes mensajes que se intercambian los puntos de acceso (APs) de la red y el controlador se comienza a capturar el tráfico justo antes de la ejecución del segundo comando. Se ejecuta la siguiente orden en una terminal:

```
sudo wireshark
```

Seguidamente, se selecciona la interfaz de *loopback* para capturar el tráfico que intercambian el controlador y los puntos de acceso.

Al comienzo de la captura se generan y envían desde cada uno de los APs mensajes hacia el controlador siguiendo un orden específico para el establecimiento de la conexión. Si se observa la captura de tráfico, se podrá comprobar que ningún AP inicia la conexión sin que el anterior haya finalizado. El procedimiento es el siguiente:

1. El punto de acceso inicia el establecimiento de la conexión TCP con el controlador.
2. A continuación, el controlador envía un mensaje OFPT_HELLO al punto de acceso y este le contesta con otro mensaje de este tipo.
3. La configuración continúa con la solicitud del detalle de características del controlador al punto de acceso mediante el envío del paquete OFPT_FEATURES_REQUEST.
4. Entre la solicitud anterior y la respuesta, el AP envía varios mensajes OFPT_SET_CONFIG y OFPT_PORT_STATUS con alguna información del estado del puerto.
5. Finalmente, el AP indica en un paquete OFPT_FEATURES_REPLY datos de interés para que el controlador los almacene. Algunos de estos datos son su dirección MAC local, el número de puertos que tiene activos, la dirección MAC de cada uno de estos puertos, si los enlaces son físicos o inalámbricos, cuál es la velocidad soportada, etc.

La Figura 4.2 muestra el intercambio de mensajes en la fase de establecimiento de la conexión entre el punto de acceso ap1 y el controlador. El mensaje más interesante de todos los indicados es en el que el AP le indica al controlador de la red las opciones que soporta. En las Figuras 4.3 y 4.4 se muestra lo que incluye el mensaje OFPT_FEATURES_REPLY de ap1 en la red diseñada.

De la Figura 4.3 se recoge que ap1 puede ejecutar todas las acciones posibles soportadas en la versión de OpenFlow que está utilizando, y que tiene prácticamente todas las estadísticas activas. Además, en ella se puede ver que no es capaz de controlar los posibles bucles que se formen en la red, lo que va a resultar de especial interés en el análisis de la LAN 2. También de ella se extrae el número de puerto TCP que conectará a ap1 con el controlador de la red, siendo esta información de vital importancia debido a que el intercambio de los paquetes entre el controlador y cada uno de los APs se realiza a través de la dirección de *localhost* (127.0.0.1),

| | | | | |
|-------------|-----------|-----------|----------|--------------------------------------------------------------------------------------------------------------|
| 51 4.9457.. | 127.0.0.1 | 127.0.0.1 | TCP | 74 35554 → 6653 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=403181099 TSecr=0 WS=512 |
| 52 4.9457.. | 127.0.0.1 | 127.0.0.1 | TCP | 74 6653 → 35554 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=403181099 TSecr=403181106 |
| 53 4.9457.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 35554 → 6653 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=403181099 TSecr=403181106 |
| 54 4.9459.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 74 Type: OFPT_HELLO Seq=1 Ack=9 Win=44032 Len=0 TSval=403181101 TSecr=403181106 |
| 55 4.9469.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 35554 → 6653 [ACK] Seq=1 Ack=9 Win=44032 Len=0 TSval=403181101 TSecr=403181106 |
| 56 4.9481.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 74 Type: OFPT_HELLO Seq=9 Ack=9 Win=44032 Len=0 TSval=403181102 TSecr=403181102 |
| 57 4.9481.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 6653 → 35554 [ACK] Seq=9 Ack=9 Win=44032 Len=0 TSval=403181102 TSecr=403181102 |
| 58 4.9484.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 74 Type: OFPT_FEATURES_REQUEST Seq=9 Ack=17 Win=44032 Len=0 TSval=403181102 TSecr=403181102 |
| 59 4.9484.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 35554 → 6653 [ACK] Seq=9 Ack=17 Win=44032 Len=0 TSval=403181102 TSecr=403181102 |
| 60 4.9484.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 78 Type: OFPT_SET_CONFIG Seq=9 Ack=29 Win=44032 Len=0 TSval=403181102 TSecr=403181102 |
| 61 4.9485.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 35554 → 6653 [ACK] Seq=9 Ack=29 Win=44032 Len=0 TSval=403181102 TSecr=403181102 |
| 62 4.9501.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 130 Type: OFPT_PORT_STATUS Seq=29 Ack=73 Win=44032 Len=0 TSval=403181104 TSecr=403181104 |
| 63 4.9501.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 6653 → 35554 [ACK] Seq=29 Ack=73 Win=44032 Len=0 TSval=403181104 TSecr=403181104 |
| 64 4.9501.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 130 Type: OFPT_PORT_STATUS Seq=29 Ack=137 Win=44032 Len=0 TSval=403181104 TSecr=403181104 |
| 65 4.9501.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 6653 → 35554 [ACK] Seq=29 Ack=137 Win=44032 Len=0 TSval=403181104 TSecr=403181104 |
| 66 4.9502.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 130 Type: OFPT_PORT_STATUS Seq=29 Ack=201 Win=44032 Len=0 TSval=403181104 TSecr=403181104 |
| 67 4.9502.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 6653 → 35554 [ACK] Seq=29 Ack=201 Win=44032 Len=0 TSval=403181104 TSecr=403181104 |
| 68 4.9521.. | 127.0.0.1 | 127.0.0.1 | OpenFlow | 338 Type: OFPT_FEATURES_REPLY Seq=29 Ack=473 Win=44032 Len=0 TSval=403181106 TSecr=403181106 |
| 69 4.9521.. | 127.0.0.1 | 127.0.0.1 | TCP | 66 6653 → 35554 [ACK] Seq=29 Ack=473 Win=44032 Len=0 TSval=403181106 TSecr=403181106 |
| 70 4.9968.. | 127.0.0.1 | 127.0.0.1 | TCP | 74 35556 → 6653 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=403181150 TSecr=0 WS=512 |
| 71 4.9968.. | 127.0.0.1 | 127.0.0.1 | TCP | 74 6653 → 35556 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=403181150 TSecr=403181150 |

Figura 4.2: Mensajes intercambiados en la configuración de un AP.

```

▶ Transmission Control Protocol, Src Port: 35554, Dst Port: 6653, Seq: 201, Ack: 29, Len: 272
▼ OpenFlow 1.0
  .000 0001 = Version: 1.0 (0x01)
  Type: OFPT_FEATURES_REPLY (6)
  Length: 272
  Transaction ID: 4041739399
  ▼ Datapath unique ID: 0x1000000000000001
    MAC addr: Private_00:00:00 (10:00:00:00:00:00)
    Implementers part: 0x0001
    n_buffers: 0
    n_tables: 254
    Pad: 000000
  ▼ capabilities: 0x000000c7
    .....1 = Flow statistics: True
    .....1. = Table statistics: True
    .....1.. = Port statistics: True
    .....0... = Group statistics: False
    .....0. .... = Can reassemble IP fragments: False
    .....1. .... = Queue statistics: True
    .....0 ..... = Switch will block looping ports: False
  ▼ actions: 0x00000fff
    .....1 = Output to switch port: True
    .....1. = Set the 802.1q VLAN id: True
    .....1.. = Set the 802.1q priority: True
    .....1... = Strip the 802.1q header: True
    .....1.... = Ethernet source address: True
    .....1. .... = Ethernet destination address: True
    .....1. .... = IP source address: True
    .....1. .... = IP destination address: True
    .....1. .... = IP ToS (DSCP field, 6 bits): True
    .....1. .... = TCP/UDP source port: True
    .....1. .... = TCP/UDP destination port: True
    .....1... = Output to queue: True

```

Figura 4.3: Características soportadas por ap1.

```

▼ Port data 3
  Port number: 1
  HW Address: 02:00:00:00:15:00 (02:00:00:00:15:00)
  Port Name: ap1-wlan1
  ▶ Config flags: 0x00000000
  ▶ State flags: 0x00000000
  ▶ Current features: 0x00000000
  Advertised features: 0x00000000
  Features supported: 0x00000000
  Features advertised by peer: 0x00000000
▼ Port data 4
  Port number: 2
  HW Address: 02:00:00:00:16:00 (02:00:00:00:16:00)
  Port Name: ap1-wlan2
  ▶ Config flags: 0x00000000
  ▶ State flags: 0x00000001
  ▶ Current features: 0x00000000
  Advertised features: 0x00000000
  Features supported: 0x00000000
  Features advertised by peer: 0x00000000
▼ Port data 5
  Port number: 3
  HW Address: 02:00:00:00:17:00 (02:00:00:00:17:00)
  Port Name: ap1-wlan3
  ▶ Config flags: 0x00000001
  ▶ State flags: 0x00000001
  ▶ Current features: 0x00000000
  Advertised features: 0x00000000
  Features supported: 0x00000000
  Features advertised by peer: 0x00000000

```

Figura 4.4: Puertos disponibles en ap1.

y la única forma de saber con qué AP se está comunicando el controlador es distinguir el puerto TCP de cada una de esas conexiones. En la Tabla 4.1 se ha extraído de los mensajes entre el controlador y cada uno de los APs el número de puerto TCP que se usa para establecer la comunicación entre ellos.

Por último, en la Figura 4.4, ap1 indica los puertos que tiene habilitados y disponibles este punto de acceso: ap1-wlan1, ap1-wlan2 y ap1-wlan3.

| AP | ap1 | ap2 | ap3 | ap4 | ap5 | ap6 | ap7 | ap9 | ap10 | ap11 |
|--------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Número puerto TCP | 35554 | 35556 | 35558 | 35560 | 35562 | 35564 | 35566 | 35568 | 35570 | 35572 |

Tabla 4.1: Puertos TCP utilizados por los APs para comunicarse con el controlador.

4.3. Conectividad, latencia y capacidad de las comunicaciones de la red

4.3.1. Conectividad y latencia

La primera prueba que se hizo con estos elementos fue comprobar que hubiera conectividad IP entre todas las estaciones base. Para ello se empleó el comando `pingall`, que arrojó que todas ellas podían comunicarse. En total con el comando se enviaron 110 mensajes ping por la red, de los cuales no se perdió ninguno y todos recibieron respuesta. En la Tabla 4.2 se recogen los valores de latencia obtenidos en este examen a la red.

| Nodos | sta1 | sta2 | sta3 | sta4 | sta5 | sta6 | sta7 | sta8 | sta9 | sta10 | sta11 |
|-------|-------|-------|------|------|------|------|------|------|------|-------|-------|
| sta1 | - | 9,21 | 9,37 | 127 | 85,8 | 80,1 | 71,6 | 55,2 | 60,7 | 83 | 77,3 |
| sta2 | 2,33 | - | 6,52 | 78,5 | 75,5 | 84 | 71,7 | 54,1 | 67,5 | 75,9 | 65,1 |
| sta3 | 0,834 | 0,801 | - | 77,1 | 77,3 | 91,6 | 63 | 58,8 | 59,7 | 80,6 | 70,5 |
| sta4 | 25,6 | 32 | 22,9 | - | 45,6 | 57,4 | 41,8 | 65 | 87,7 | 100 | 78,4 |
| sta5 | 39,1 | 31,3 | 31,1 | 14 | - | 42,4 | 50,2 | 92 | 89 | 109 | 88,1 |
| sta6 | 36,9 | 34,6 | 37,7 | 26,5 | 21,8 | - | 76,3 | 86,4 | 98,1 | 99,4 | 87,4 |
| sta7 | 34,6 | 52,3 | 34,7 | 13,5 | 29,1 | 42,4 | - | 72,2 | 84,5 | 93,1 | 82,2 |
| sta8 | 25,4 | 25,8 | 15,3 | 27 | 37,8 | 37,1 | 30,7 | - | 48,3 | 54,6 | 43,5 |
| sta9 | 21,7 | 20,5 | 29,3 | 36,7 | 48,9 | 54,6 | 56,6 | 15,6 | - | 51,5 | 1,72 |
| sta10 | 50 | 32,6 | 27 | 37,6 | 41,9 | 53,6 | 41,7 | 22,2 | 24,3 | - | 90,4 |
| sta11 | 46,4 | 26,4 | 27 | 30 | 49,2 | 93,6 | 54,9 | 18,6 | 3,94 | 14,2 | - |

Tabla 4.2: Latencias en milisegundos (ms) obtenidas tras el test de conectividad de la red.

La Tabla 4.2 clarifica el valor que tienen las cachés y tablas de flujos dentro del encaminamiento de la red y en el funcionamiento de los diferentes equipos de la misma y sus conexiones. En este caso, si se pone interés en la latencia existente en la comunicación entre sta1 y sta3 y entre sta3 y sta1, resulta que esta última no llega a ser ni una décima parte de la primera. La explicación está precisamente en la utilidad que tienen los flujos que instala el controlador en cada uno de los APs para que sean capaces de encaminar el tráfico sin necesidad de consultarle. Tal y como se indicó en la exposición teórica del Capítulo 2, estas entradas almacenadas en las tablas de flujos no tienen vigencia indefinida, sino que debido a razones de peso en la red estos datos de las tablas suelen tener asociada una fecha de validez, y a su término serán eliminados de las mismas. Algunos de los motivos que justifican este borrado son que estas entradas pueden dejar de usarse o incluso ser erróneas con el cambio en la configuración de la red, debido a que

la red SDN debe permitir cambios y adaptarse a la situación en cada momento determinado.

4.3.2. Capacidad y ancho de banda

Una vez comprobado que existía plena comunicación entre todos los nodos de la red, se examinó la velocidad de las conexiones entre ellos. Para ello se estableció a *sta1* como nodo de referencia del que partirían todos los exámenes de la red. Los datos de ancho de banda obtenidos son los detallados en la Tabla 4.3.

Durante el examen se ejecutó en el terminal el siguiente comando para cada una de las estaciones:

```
iperf sta1 staX
```

| Origen: sta1 | sta2 | sta3 | sta4 | sta5 | sta6 | sta7 | sta8 | sta9 | sta10 | sta11 |
|--------------|------|------|------|------|------|------|------|------|-------|-------|
| UL | 7,11 | 6,14 | 5,91 | 5,98 | 5,9 | 5,96 | 3,76 | 4,28 | 4,22 | 4,15 |
| DL | 7,64 | 6,7 | 6,55 | 6,42 | 6,32 | 6,47 | 4,24 | 4,58 | 4,68 | 4,48 |

Tabla 4.3: Capacidad de las conexiones entre los nodos de la red en Mbps.

En la tabla, la fila UL indica la velocidad disponible en la conexión desde *sta1* hacia la estación destino, y la fila DL la correspondiente al enlace inverso.

Los resultados de este test de capacidad de la red han arrojado que la velocidad de las conexiones entre los nodos que conforman la red suele tener valores entre los 4Mbps y los 8Mbps. No obstante, cuantos menos saltos intermedios haya entre las estaciones base objeto de la comprobación, mayor es el rendimiento de ancho de banda disponible que presentan los diferentes valores obtenidos. Pueden deberse a motivos como la distancia entre nodos, que provoca atenuación y pérdidas, o en retardos ocasionados en la parte WAN de la red debido a comunicaciones entre los APs y el controlador. Si se cambia el exponente de propagación a uno de menor atenuación, el ancho de banda disponible y, por tanto, la velocidad de cada uno de los enlaces aumenta. En caso de que se aumentara el exponente del modelo de propagación, la tasa disponible debería reducirse.

4.4. Análisis de las diferentes áreas de la red

Como se indicó en el Apartado 4.1 del presente capítulo, la red diseñada cuenta con varias zonas o áreas cada una con funciones y protocolos específicos. A continuación en este apartado

se detallan las características y el funcionamiento de cada una de ellas.

4.4.1. WAN

La red de área extensa de la topología de la Figura 4.1 está formada por tres APs que se comunican entre ellos mediante enlaces inalámbricos configurados como elementos mesh. Además, cada uno de estos enlaces se conecta a las diferentes LANs que componen la red. Precisamente su utilidad es esta, interconectar a todos los nodos de la red para que pueda existir comunicación entre ellos, si fuera necesario.

Estos tres puntos de acceso se comunican por el medio inalámbrico dentro del canal 1. Cada uno de ellos cuenta con tres WLANS que les sirven para establecer tres conexiones inalámbricas con equipos de la red. Una de estas WLANS de cada AP se utiliza para establecer una red mesh que permita comunicarlos sin necesidad de crear tres enlaces diferentes. En resumen, se consigue ahorrar recursos, pues en lugar de tener que establecer tres conexiones (ap1-ap2, ap2-ap3, ap3-ap1) teniendo cada AP que aportar dos recursos inalámbricos, se emplean las características de las redes mesh para interconectar todos los equipos de esa parte WAN de la red usando un único recurso por AP.

En el script indicado, las líneas de código que se corresponden con estas funcionalidades son las que se indican a continuación.

```
ap1 = net.addAccessPoint('ap1', wlans=3, ssid='ssid1,', position='20,100,0',  
channel='1')  
  
ap2 = net.addAccessPoint('ap2', wlans=3, ssid='ssid2,', position='40,100,0',  
channel='1')  
  
ap3 = net.addAccessPoint('ap3', wlans=3, ssid='ssid3,', position='40,80,0',  
channel='1')
```

No hay que olvidar que dentro de esta red todos los APs tienen comunicación directa con el controlador de la red. Esto quiere decir que la función de los tres puntos de acceso de la zona WAN únicamente sirven de interconexión entre las LANs como se ha indicado, es decir, cuando un paquete sale dirigido de sta10 hacia sta11, no tiene que llegar a ap1, ap2 ni ap3 para que sea encaminado hacia el nodo destino sta11, sino que ap10 y ap9 consultarán al controlador lo que deben hacer con ese tráfico y este les indicará que lo envíen de la manera más directa posible a sta11. En este caso la ruta indicada tendría como saltos intermedios a ap10, ap9 y llegaría directamente a sta11. Este proceso será expuesto en el análisis realizado para la LAN 3, en el

apartado 4.4.4 del presente capítulo.

4.4.2. LAN 1

Posiblemente sea la más sencilla de las tres LANs de la red. Consta de tres estaciones base conectadas al punto de acceso ap1. La función de este AP dentro de la red es comunicar a los tres nodos emplazados, sta1, sta2 y sta3.

Pero, ¿qué ocurre cuando se ejecuta un mensaje ping entre todos los nodos? Sucede algo novedoso: en el primer ping de cada una de las estaciones sí que se generan mensajes OpenFlow entre el controlador y ap1. Sin embargo, si se vuelve a repetir el ping entre las estaciones no se generan nuevos paquetes entre ellos. Esto se debe a que, aunque el controlador no ha indicado que se instalen nuevos flujos en el AP y, por tanto, la tabla que los contiene se encuentra vacía, ap1 es capaz de comportarse como un switch normal y encaminar el tráfico de la LAN 1.

Como en cualquier otro escenario de redes, sta1, sta2 y sta3 almacenan en sus cachés ARP las direcciones IP y MAC de los nodos con los que han conectado, así como la interfaz por la que debe salir este tráfico. En esta parte tan sencilla de la red no parece de mucha utilidad la presencia de la interfaz en la caché de ARP, pero cuando se analizan redes más complejas sí que es útil, ya que proporciona información acerca del encaminamiento del tráfico. En las Figuras 4.5, 4.6 y 4.7 se puede ver el contenido de las cachés ARP de las tres estaciones.

| Dirección | TipoHW | DirecciónHW | Indic | Máscara | Interfaz |
|-----------|--------|-------------------|-------|---------|------------|
| 10.0.0.3 | ether | 00:00:00:00:00:03 | C | | sta1-wlan0 |
| 10.0.0.2 | ether | 00:00:00:00:00:02 | C | | sta1-wlan0 |

Figura 4.5: Caché ARP de sta1.

| Dirección | TipoHW | DirecciónHW | Indic | Máscara | Interfaz |
|-----------|--------|-------------------|-------|---------|------------|
| 10.0.0.3 | ether | 00:00:00:00:00:03 | C | | sta2-wlan0 |
| 10.0.0.1 | ether | 00:00:00:00:00:01 | C | | sta2-wlan0 |

Figura 4.6: Caché ARP de sta2.

| Dirección | TipoHW | DirecciónHW | Indic | Máscara | Interfaz |
|-----------|--------|-------------------|-------|---------|------------|
| 10.0.0.1 | ether | 00:00:00:00:00:01 | C | | sta3-wlan0 |
| 10.0.0.2 | ether | 00:00:00:00:00:02 | C | | sta3-wlan0 |

Figura 4.7: Caché ARP de sta3.

4.4.3. LAN 2

Esta sección de la red diseñada es un poco más especial que el resto. En ella hay emplazado un anillo formado por cuatro puntos de acceso a los que se conectan cuatro nodos. En un principio se configuraron estos APs en el script como se haría con puntos de acceso habituales. Sin embargo, cuando se lanzaba el examen de conectividad a la red era imposible obtener un 100 % de mensajes recibidos porque nunca se podía acceder a los nodos de la LAN 2 desde dentro ni desde fuera de ella. El controlador bloqueaba estas comunicaciones para evitar que los mensajes ARP request entraran al anillo y generaran una tormenta de *broadcast*, dado que estos mensajes aún tratándose de redes SDN se transmiten por inundación.

En ese escenario era imposible establecer comunicación y conectividad totales en la red, por lo que hubo que incluir un nuevo parámetro en la configuración de estos cuatro puntos de acceso en el script. Se trata de activar la opción STP en ellos. El protocolo *Spanning Tree Protocol* (STP) resuelve los casos de redundancia en una red en la que existe el peligro de que se formen bucles o *loops* en los switches y puntos de acceso. En este caso, ese era el problema que se presentaba en la topología diseñada para la LAN 2.

Para activar la opción con STP en la red hay que ejecutar la siguiente orden en un terminal:

```
sudo python topo.py -s
```

Activando la función STP de los APs de la red que se encuentran en el anillo se logró acceder a ella desde cualquier punto de la misma. El protocolo STP, en resumen, envía una serie de mensajes de configuración en los que aprende cómo es la topología de la red. Durante el proceso de aprendizaje, los puertos de los switches y puntos de acceso aparecen etiquetados como STP_LEARN y solo reciben y envían los tráfico de configuración STP. Pasados unos segundos y tras conocer la topología de la red, el protocolo STP configura un switch o AP raíz, que será el encargado de ser la salida del bucle hacia la red. Además, se bloquea uno de los puertos del anillo con el fin de que por ese puerto no se envíe ni reciba tráfico. Al puerto bloqueado se le asigna el estado *STP_BLOCK*. A través del resto de interfaces se pueden transmitir los paquetes de la red, ya sin peligro a que debido a inundación se genere una tormenta de *broadcast*.

Las interfaces de los puntos de acceso pasan durante el proceso de aprendizaje y resolución de STP por varios estados, en primer lugar son puertos de aprendizaje (Figura 4.8), para después pasar a reenviar paquetes (Figura 4.9) o a estar bloqueado (Figura 4.10). En la topología estudiada es ap6 el que configurará uno de sus puertos bloqueado para romper el bucle.

Para que este proceso pueda suceder en la red, es necesario configurar los APs en el script

```

mininet-wifi> sh ovs-ofctl show ap4
OFP_T_FEATURES_REPLY (xid=0x2): dpid:1000000000000004
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_d
st mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(ap4-wlan1): addr:00:00:00:00:04:00
  config: 0
  state: STP_LEARN
  speed: 0 Mbps now, 0 Mbps max
2(ap4-eth2): addr:16:df:2f:b9:a8:6c
  config: 0
  state: STP_LEARN
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
3(ap4-eth3): addr:a2:a1:75:c1:24:92
  config: 0
  state: STP_LEARN
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
4(ap4-eth4): addr:52:3a:b9:b4:ad:12
  config: 0
  state: STP_LEARN
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
LOCAL(ap4): addr:00:00:00:00:04:00
  config: PORT_DOWN
  state: LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max
OFP_T_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

```

Figura 4.8: Estado inicial STP en los puertos de ap4.

```

mininet-wifi> sh ovs-ofctl show ap4
OFP_T_FEATURES_REPLY (xid=0x2): dpid:1000000000000004
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_d
st mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(ap4-wlan1): addr:00:00:00:00:04:00
  config: 0
  state: STP_FORWARD
  speed: 0 Mbps now, 0 Mbps max
2(ap4-eth2): addr:16:df:2f:b9:a8:6c
  config: 0
  state: STP_FORWARD
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
3(ap4-eth3): addr:a2:a1:75:c1:24:92
  config: 0
  state: STP_FORWARD
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
4(ap4-eth4): addr:52:3a:b9:b4:ad:12
  config: 0
  state: STP_FORWARD
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
LOCAL(ap4): addr:00:00:00:00:04:00
  config: PORT_DOWN
  state: LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max

```

Figura 4.9: Estado STP en los puertos de ap4: STP_FORWARD.

```

mininet-wifi> sh ovs-ofctl show ap6
OFPST_FEATURES_REPLY (xid=0x2): dpid:1000000000000006
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_d
st mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(ap6-wlan1): addr:00:00:00:00:06:00
  config: 0
  state: STP_FORWARD
  speed: 0 Mbps now, 0 Mbps max
2(ap6-eth2): addr:d2:a9:2e:87:5a:05
  config: 0
  state: STP_FORWARD
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
3(ap6-eth3): addr:0a:35:d9:98:4f:f7
  config: 0
  state: STP_BLOCK
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
LOCAL(ap6): addr:00:00:00:00:06:00
  config: PORT_DOWN
  state: LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max

```

Figura 4.10: Estado STP en los puertos de ap6: uno de ellos STP_BLOCK.

de la siguiente forma:

```

ap4 = net.addAccessPoint('ap4', ssid='new-ssid4', mode='g', channel='1',
failMode="standalone", position='100,100,0', stp=True)

```

4.4.4. LAN 3

Esta última sección de la red está compuesta por tres puntos de acceso y cuatro nodos diferentes. En este caso, sta9 se conecta a ap9, sta10 tiene conexión inalámbrica con ap10, sta11 con ap11 y sta8 se comunica directamente con ap3. Sin duda se trata de la topología más especial de la red diseñada, pues no es intuitivo a primera vista lo que ocurriría si un paquete de sta10 se quisiera enviar a sta11. Este es el objetivo deseado en este área de la red diseñada, comprobar que el controlador ordena la instalación de flujos adecuada en los puntos de acceso para permitir la comunicación de dos nodos de una misma zona en la que hay varios puntos de acceso.

Llegados a este punto es necesario matizar, como se indicó en anteriores análisis, que el controlador casi siempre instala nuevos flujos en los APs, excepto cuando se trata de paquetes ARP request. Dado que este tipo de tráfico debe ser reenviado por todos los puertos de los switches y puntos de acceso de la red, cada vez que es necesario encaminar un paquete de estas características que se propaga por el mecanismo de inundación puede generar graves problemas de consumo de recursos en la red, por lo que la decisión de si se transmite o no la debe tomar el controlador, y no un switch en base a entradas en su tabla de flujos que no tiene conocimiento alguno del funcionamiento y comportamiento de la red en ese momento.

Como consecuencia, para la resolución de esta cuestión se toma como ejemplo la respuesta ARP (mensaje ARP reply) de sta11 a sta10, debido a que los paquetes que envía sta10 a sta11 van dirigidos a la dirección de *broadcast*, se reenvían por todos los puertos de los switches y APs y no se instalan flujos. Sin embargo, con el paquete ARP reply de sta11 a sta10 es más sencillo entender qué ocurre en la red. Cuando este mensaje es enviado por sta11 directamente va hacia ap11, la única salida que tiene ap11 hacia la red se encuentra en ap9, el controlador indicará a ap11 que instale un flujo nuevo en la tabla y envíe el paquete por su interfaz con ap9. Una vez que llega a este punto de acceso se plantea la duda de qué camino seguirá el tráfico de sta11. Según las propiedades de las redes SDN, el paquete debería ser encaminado hacia ap10 mediante la instalación de un flujo nuevo en la tabla de flujos de ap9, sin necesidad de pasar por la WAN porque todos los switches y puntos de acceso de una red SDN tienen comunicación directa con el controlador de la red por la interfaz de *loopback*. Cuando el paquete llegue a ap10, el controlador le indicará también a través de la instalación de un flujo que debe reenviarlo hacia sta10. ¿Ocurrirá esto en la ejecución de la red? Para comprobar lo que sucede basta con realizar un ping entre las dos estaciones y capturar los mensajes intercambiados.

Para la realización de esta prueba se ha ejecutado nuevamente el escenario de Mininet WiFi y las conexiones TCP entre el controlador y los APs han cambiado. A continuación, en la Tabla 4.4 se muestran los nuevos puertos de estas comunicaciones TCP.

| AP | ap1 | ap2 | ap3 | ap4 | ap5 | ap6 | ap7 | ap9 | ap10 | ap11 |
|-------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Número puerto TCP | 59928 | 59930 | 59932 | 59934 | 59936 | 59938 | 59940 | 59942 | 59944 | 59946 |

Tabla 4.4: Puertos TCP utilizados por los APs para comunicarse con el controlador en el análisis de la LAN 3.

En las Figuras 4.11, 4.12 y 4.13 se muestran los mensajes enviados por el controlador de la red a ap11 (puerto 59946), ap9 (puerto 59942) y ap10 (puerto 59944).

Empleando el comando `net` se obtiene información de las interfaces de los elementos de la red. De todas ellas las que son de relevancia en el análisis de la LAN 3, puesto que indican qué interfaces Ethernet se comunican entre ellas, son las siguientes:

```
mininet-wifi>net
```

```
...
```

```
ap9 lo: ap9-wlan1:wifi ap9-wlan2:wifi ap9-wlan3:wifi ap9-wlan4:wifi
```

```
ap9-eth5:ap10-eth3 ap9-eth6:ap11-eth3 ap9-eth7:ap3-eth4
```

| | | | | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|----------|--------------------------------------|
| 2985 14.0... | 00:00:00_00:00:11 | 00:00:00_00:00:10 | OpenFlow | 132 Type: OFPT_PACKET_OUT |
| 2986 14.0... | 127.0.0.1 | 127.0.0.1 | TCP | 66 59946 → 6653 [ACK] Seq=22923 Ack= |
| <p> ▶ Frame 2985: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00) ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 ▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 59946, Seq: 23605, Ack: 22923, Len: 66 ▼ OpenFlow 1.0 .000 0001 = Version: 1.0 (0x01) Type: OFPT_PACKET_OUT (13) Length: 66 Transaction ID: 0 Buffer Id: 0xffffffff In port: 1 Actions length: 8 Actions type: Output to switch port (0) Action length: 8 Output port: 3 Max length: 0 ▶ Ethernet II, Src: 00:00:00_00:00:11 (00:00:00:00:00:11), Dst: 00:00:00_00:00:10 (00:00:00:00:00:10) ▼ Address Resolution Protocol (reply) Hardware type: Ethernet (1) Protocol type: IPv4 (0x0800) Hardware size: 6 Protocol size: 4 Opcode: reply (2) Sender MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11) Sender IP address: 10.0.0.11 Target MAC address: 00:00:00_00:00:10 (00:00:00:00:00:10) Target IP address: 10.0.0.10 </p> | | | | |

Figura 4.11: Paquete de controlador a ap11.

```

ap10 lo: ap10-wlan1:wifi ap10-wlan2:wifi ap10-eth3:ap9-eth5
ap11 lo: ap11-wlan1:wifi ap11-wlan2:wifi ap11-eth3:ap9-eth6
...
```

En la Figura 4.11 se muestra el mensaje OFPT_PACKET_OUT que envía el controlador a ap11 para encaminar el tráfico procedente de sta11. Se aprecia en la imagen que se instala un flujo para tráfico ARP reply, con dirección origen Ethernet 00:00:00:00:00:11 (IP 10.0.0.11), dirigido a la dirección destino Ethernet 00:00:00:00:00:10 (IP 10.0.0.10), que entra a ap11 por su puerto 1, correspondiente a la interfaz ap11-wlan1. La acción a ejecutar sobre ese tipo de tráfico que indica el controlador es su reenvío por el puerto 3 de ap11, encontrándose en él la interfaz ap11-eth3, que conecta a este punto de acceso con ap9 mediante un medio guiado.

Igualmente, las Figuras 4.12 y 4.13 contienen los otros dos flujos que ordena instalar el controlador a ap9 y ap10. En ellos los campos del flujo son los mismos que en el caso del instalado en ap11, pero cambiando las interfaces de entrada y las de salida. Para ap9 (Figura 4.12) se indica que el tráfico entrante por el puerto 6, interfaz ap9-eth6, debe reenviarse por el puerto 5, interfaz ap9-eth5, que lo conecta con ap10.

En el caso de ap10 (Figura 4.13), el tráfico ARP reply entrante por su puerto 3, interfaz ap10-eth3, debe ser reenviado por el punto de acceso por su puerto 1, correspondiente a la interfaz ap10-wlan1 que conecta con la estación sta10.

Todas estas acciones son instaladas en las tablas de flujos de los puntos de acceso, porque

| | | | | | | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|-------------------|-------------------|----------|-----|-----------------------------------|
| 2994 | 14.0... | 00:00:00_00:00:11 | 00:00:00_00:00:10 | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 2995 | 14.0... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 59942 → 6653 [ACK] Seq=23759 Ack= |
| <p>Frame 2994: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0</p> <p>Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)</p> <p>Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1</p> <p>Transmission Control Protocol, Src Port: 6653, Dst Port: 59942, Seq: 23813, Ack: 23759, Len: 66</p> <p>OpenFlow 1.0</p> <p>.000 0001 = Version: 1.0 (0x01)</p> <p>Type: OFPT_PACKET_OUT (13)</p> <p>Length: 66</p> <p>Transaction ID: 0</p> <p>Buffer Id: 0xffffffff</p> <p>In port: 6</p> <p>Actions length: 8</p> <p>Actions type: Output to switch port (0)</p> <p>Action length: 8</p> <p>Output port: 5</p> <p>Max length: 0</p> <p>Ethernet II, Src: 00:00:00_00:00:11 (00:00:00:00:00:11), Dst: 00:00:00_00:00:10 (00:00:00:00:00:10)</p> <p>Address Resolution Protocol (reply)</p> <p>Hardware type: Ethernet (1)</p> <p>Protocol type: IPv4 (0x0800)</p> <p>Hardware size: 6</p> <p>Protocol size: 4</p> <p>Opcode: reply (2)</p> <p>Sender MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11)</p> <p>Sender IP address: 10.0.0.11</p> <p>Target MAC address: 00:00:00_00:00:10 (00:00:00:00:00:10)</p> <p>Target IP address: 10.0.0.10</p> | | | | | | |

Figura 4.12: Paquete de controlador a ap9.

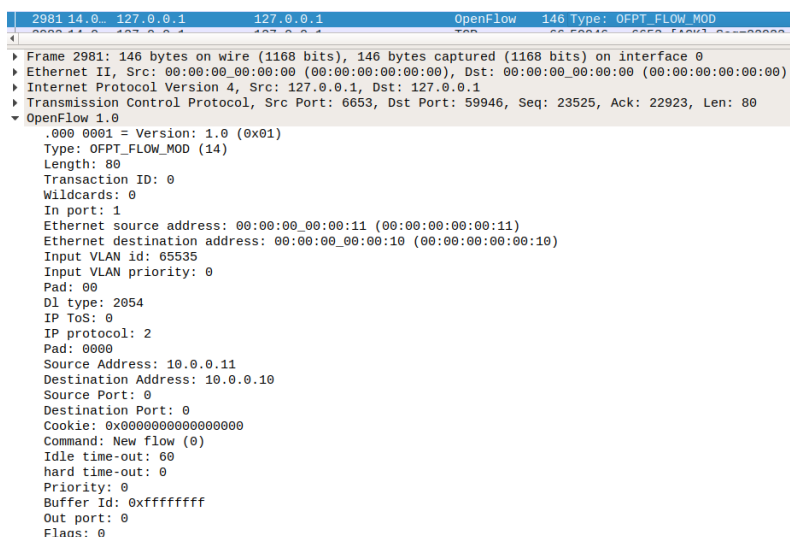
| | | | | | | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|-------------------|-------------------|----------|-----|-----------------------------------|
| 2999 | 14.0... | 00:00:00_00:00:11 | 00:00:00_00:00:10 | OpenFlow | 132 | Type: OFPT_PACKET_OUT |
| 3000 | 14.0... | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 59944 → 6653 [ACK] Seq=23535 Ack= |
| <p>Frame 2999: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0</p> <p>Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)</p> <p>Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1</p> <p>Transmission Control Protocol, Src Port: 6653, Dst Port: 59944, Seq: 23833, Ack: 23535, Len: 66</p> <p>OpenFlow 1.0</p> <p>.000 0001 = Version: 1.0 (0x01)</p> <p>Type: OFPT_PACKET_OUT (13)</p> <p>Length: 66</p> <p>Transaction ID: 0</p> <p>Buffer Id: 0xffffffff</p> <p>In port: 3</p> <p>Actions length: 8</p> <p>Actions type: Output to switch port (0)</p> <p>Action length: 8</p> <p>Output port: 1</p> <p>Max length: 0</p> <p>Ethernet II, Src: 00:00:00_00:00:11 (00:00:00:00:00:11), Dst: 00:00:00_00:00:10 (00:00:00:00:00:10)</p> <p>Address Resolution Protocol (reply)</p> <p>Hardware type: Ethernet (1)</p> <p>Protocol type: IPv4 (0x0800)</p> <p>Hardware size: 6</p> <p>Protocol size: 4</p> <p>Opcode: reply (2)</p> <p>Sender MAC address: 00:00:00_00:00:11 (00:00:00:00:00:11)</p> <p>Sender IP address: 10.0.0.11</p> <p>Target MAC address: 00:00:00_00:00:10 (00:00:00:00:00:10)</p> <p>Target IP address: 10.0.0.10</p> | | | | | | |

Figura 4.13: Paquete del controlador a ap10.

previamente el controlador indicó a los APs que incluyeran estas entradas en las tablas mediante mensajes OFPT_FLOW_MOD. En la Figura 4.14 se encuentra a modo de ejemplo el contenido del mensaje que envía el controlador a ap11 para que instale el flujo mostrado en la Figura 4.11.

4.5. Resultados obtenidos

Una vez analizado el comportamiento de la red, se puede asegurar que ha satisfecho todas las pruebas que se han realizado sobre ella. A continuación se indica brevemente cada una de ellas:



```

2981 14.0. 127.0.0.1 127.0.0.1 OpenFlow 146 Type: OFPT_FLOW_MOD
    ...
    ▶ Frame 2981: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
    ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
    ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
    ▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 59946, Seq: 23525, Ack: 22923, Len: 80
    ▼ OpenFlow 1.0
      .000 0001 = Version: 1.0 (0x01)
      Type: OFPT_FLOW_MOD (14)
      Length: 80
      Transaction ID: 0
      Wildcards: 0
      In port: 1
      Ethernet source address: 00:00:00_00:00:11 (00:00:00:00:00:11)
      Ethernet destination address: 00:00:00_00:00:10 (00:00:00:00:00:10)
      Input VLAN id: 65535
      Input VLAN priority: 0
      Pad: 00
      DL type: 2054
      IP ToS: 0
      IP protocol: 2
      Pad: 0000
      Source Address: 10.0.0.11
      Destination Address: 10.0.0.10
      Source Port: 0
      Destination Port: 0
      Cookie: 0x0000000000000000
      Command: New flow (0)
      Idle time-out: 60
      hard time-out: 0
      Priority: 0
      Buffer Id: 0xffffffff
      Out port: 0
      Flags: 0
  
```

Figura 4.14: Paquete OFPT_FLOW_MOD del controlador a ap11.

- Test de conectividad entre todos los nodos de la red, ejecutado mediante el comando pingall. Además, durante este chequeo también se confirmó la utilidad de las cachés, pues la latencia disminuía en la segunda comunicación entre diferentes nodos.
- La red proporciona un ancho de banda estable en sus comunicaciones, entre los 4 y 8 Mbps.
- El controlador es eficaz en el encaminamiento de tráfico en topologías sencillas y más complejas, como la LAN 3. No se generan flujos innecesarios en las tablas ni tampoco el tráfico se distribuye por zonas en las que no debe circular.
- El controlador SDN es compatible con switches con configuración de puertos mesh, economizando así el número de recursos necesarios en las comunicaciones.
- El controlador también puede soportar al mismo tiempo enlaces inalámbricos y cableados, encaminando adecuadamente el tráfico en cada caso y permitiendo rutas mixtas.
- Aunque la red haya presentado fallos en la comunicación debido a que no hacía una adecuada gestión para controlar los bucles y anulaba completamente los puntos de acceso de la LAN 2, cuando se configuró STP en ellos respondió adecuadamente compatibilizando sus funciones con las de este protocolo.

Capítulo 5

Conclusiones y líneas futuras

En este capítulo se van a recoger las conclusiones a las que se ha llegado tras la realización de este trabajo, así como otros escenarios posibles para poder analizar de cara a nuevos proyectos.

5.1. Conclusiones

El objetivo de este trabajo es, como ya se indicó en la introducción del mismo, verificar el adecuado funcionamiento de las redes SDN, en concreto de WSDN, en escenarios con dispositivos de red inalámbricos. Tras la introducción teórica que se hizo en el primer capítulo a las redes definidas por software (SDN), al protocolo OpenFlow, estándar usado en este tipo de redes, y al emulador Mininet WiFi, se desarrollaron dos capítulos cuya finalidad no era más que conocer cómo funciona el intercambio de flujos entre controlador y los dispositivos de reenvío de la red (Capítulo 3), y verificar que en una red customizada se cumplen las premisas características de este tipo de redes (Capítulo 4).

Los fundamentos teóricos en los que se enmarca este trabajo son el funcionamiento de las redes SDN y el protocolo OpenFlow. Una red SDN se caracteriza principalmente por la presencia de un controlador que es el que se encarga de la toma de decisiones de gestión de la red. A esto se conoce como el plano de control. Por otro lado, en la red también hay otros elementos, los switches o puntos de acceso (*access points*) que se encargan de realizar las funciones de encaminamiento de tráfico de los paquetes, en función de las reglas indicadas por el plano de control para conseguir el comportamiento deseado en la red. A esta parte se le conoce el plano de datos. El protocolo OpenFlow constituye protocolo para el intercambio de mensajes entre

los dispositivos de red y el controlador.

Para estudiar ese tipo de redes se ha empleado el emulador Mininet WiFi. Es una evolución de Mininet, con la novedad de que permite crear topologías de red completamente inalámbricas basadas en SDN. Las redes WSDN tienen las mismas características que las SDN, con la peculiaridad de que los elementos encargados del plano de datos son los puntos de acceso o *access points*, pero comparten modo de funcionamiento y comunicación con las redes definidas por software.

La red analizada en el Capítulo 3 representa una red mesh en la que los APs se comunican el uno con el otro para encaminar el tráfico desde sta1 hacia sta2. El objetivo del análisis de este tipo de red es comprender el funcionamiento del intercambio de flujos entre controlador y puntos de acceso de la misma, porque esa es la principal diferencia que tienen las redes SDN respecto a las redes tradicionales. Además, la red resultante es puramente inalámbrica, por lo que también sirve para demostrar que los principios de SDN se extienden en WSDN. Todo lo analizado en ese capítulo permitió poder crear y estudiar una topología más compleja en el siguiente capítulo.

En el Capítulo 4 se estudia una red formada por tres áreas locales (LANs) unidas entre ellas por una red de área extensa (WAN) en la que los puntos de acceso establecen entre ellos una red mesh. La red diseñada tiene la finalidad de estudiar que se puede establecer y encaminar el tráfico que pasa por diferentes APs sin que sea necesario predefinir el comportamiento que se quiere en ella, es el controlador el que se encarga de esto. La LAN 1 está formada por tres estaciones base que se conectan directamente de manera inalámbrica a ap1. La LAN 2 tiene cuatro estaciones base conectadas cada una de ellas a un punto de acceso;; los puntos de acceso se conectan entre ellos en forma de anillo, una topología muy usada en las redes tradicionales. Por último, la LAN 3 tiene una sencilla topología de árbol en la que los nodos necesitan pasar por varios puntos de acceso para poder comunicarse con otras LANs, pero sin necesitar hacerlo para comunicarse entre ellos.

En el estudio de la red del Capítulo 3 se descubre que el controlador es la inteligencia de la red. Cada vez que llega tráfico a un punto de acceso y este no encuentra en su tabla de flujos cómo ha de encaminarlo, encapsula el tráfico en un paquete OFPT_PACKET_IN que envía al controlador para conocer qué debe hacer con ese tipo de tráfico. Cuando el controlador lo recibe, decide el camino o recorrido que debe seguir el paquete para llegar al destino final y responde con un mensaje OFPT_PACKET_OUT, en el que indica qué acción debe ejecutar el punto de acceso con ese paquete. Además, si se trata de tráfico cuya dirección destino no es la de *broadcast*, el controlador envía un paquete OFPT_FLOW_MOD antes del OFPT_PACKET_OUT

para ordenar al punto de acceso que debe instalar una nueva entrada en su tabla de flujos en la que incluya la acción que le indica en este último. Como la red tiene dos puntos de acceso, el estudio se hizo para cada uno de ellos, y se reveló que tenían exactamente las mismas reglas para el mismo tipo de tráfico, pero con acciones diferentes. Del análisis de los flujos se consiguió construir el esquema de interfaces de la red, pues a medida que llegaban las reglas a las tablas de los puntos de acceso se completaba la información que faltaba respecto a las conexiones entre dichas interfaces.

Una vez conocido el funcionamiento de una red WSDN sencilla, se pasó al diseño de otra más avanzada en el Capítulo 4, con diferentes interconexiones como se ha indicado anteriormente. De ella se obtienen varias conclusiones en función de la zona de la red en la que se ponga el foco. La parte interesante de esta topología es comprobar que una red definida por software puede convivir sin problemas de comunicación con conexiones físicas e inalámbricas, albergando topologías sencillas, en árbol y en bucle, así como con protocolos y configuraciones diferentes, como por ejemplo STP y mesh. Las conclusiones extraídas del capítulo son las siguientes:

- En la red de área extensa (WAN) formada por los tres puntos de acceso, al establecerse la comunicación entre ellos mediante principios de red mesh, se ahorra el número de recursos necesarios para que se conecten entre ellos. Si se hubiera empleado una conexión física cableada para conectarlos, se habrían necesitado dos interfaces por elemento, con un total de seis interfaces empleadas en la comunicación. Al configurarse como red mesh, fue necesaria una única interfaz inalámbrica para cada uno de ellos, reduciéndose a tres los recursos totales necesarios para la comunicación.
- Para que se comuniquen los nodos de la LAN 1, basta con que su tráfico llegue a ap1, porque es el punto de acceso al que se conectan directamente todas ellas. Por tanto, el tráfico de este área no pasa a otras zonas de la red como la WAN o la LAN 2, lo que permite un aprovechamiento más eficiente de los recursos, ya que no se consume ancho de banda en zonas por las que no debe ir el tráfico entre esos nodos.
- En la configuración y diseño de la LAN 2 se encontraron varios problemas debido a que al haber una topología en anillo el controlador no permitía el tráfico hacia ese área ni dentro de ese área, puesto que un simple paquete dirigido a la dirección de *broadcast* podía llegar a congestionar toda la red. Por ello, se optó por habilitar el protocolo STP de control de bucles en los cuatro puntos de acceso de esa zona de la red: ap4, ap5, ap6 y ap7. Al ejecutar la red, el protocolo se toma unos segundos para enviar paquetes en las

interfaces de los elementos que lo tienen activado y bloquea uno de los puertos en uno de estos elementos con la finalidad de romper el bucle y evitar que se puedan producir tormentas de *broadcast*. Una vez configurados los puertos de la LAN 2 con este protocolo, el controlador sí que permite el intercambio de tráfico entre los nodos de esa LAN y con otros de fuera de ella, demostrándose que, aunque el controlador tenga algunos fallos de control sobre la red, siempre se pueden compatibilizar sus funciones con otros protocolos que se emplean en redes tradicionales.

- Por último, la LAN 3 muestra que para establecer comunicación entre dos nodos emplazados en forma de árbol basta con que el controlador decida cómo se encamina el tráfico. En esta parte de la red se corrobora que verdaderamente el controlador cumple con sus funciones al tener una visión general que permite la toma de decisiones concretas en topologías de red más complejas.

Una vez mencionadas los puntos anteriores acerca de los análisis realizados en los capítulos del trabajo, hay que concluir indicando que efectivamente se ha comprobado el funcionamiento de las redes definidas por software. Aunque son un poco complejas porque se alejan de la realidad conocida hasta ahora de las redes tradicionales, se ha verificado que pueden integrarse y convivir con protocolos empleados en las tradicionales, además de con configuraciones y topologías distintas.

5.2. Diagrama de Gantt

Durante la ejecución del trabajo se han desempeñado las siguientes tareas:

1. Estudio de SDN.
2. Estudio de OpenFlow.
3. Estudio de Mininet WiFi.
4. Ejecución y estudio de escenarios predefinidos en Mininet WiFi.
5. Análisis del escenario simple meshAP en Mininet WiFi.
6. Definición y análisis de escenario complejo en Mininet WiFi.
7. Realización de la memoria del trabajo.

En la siguiente figura se indica el diagrama de Gantt de planificación y organización temporal de estas tareas.

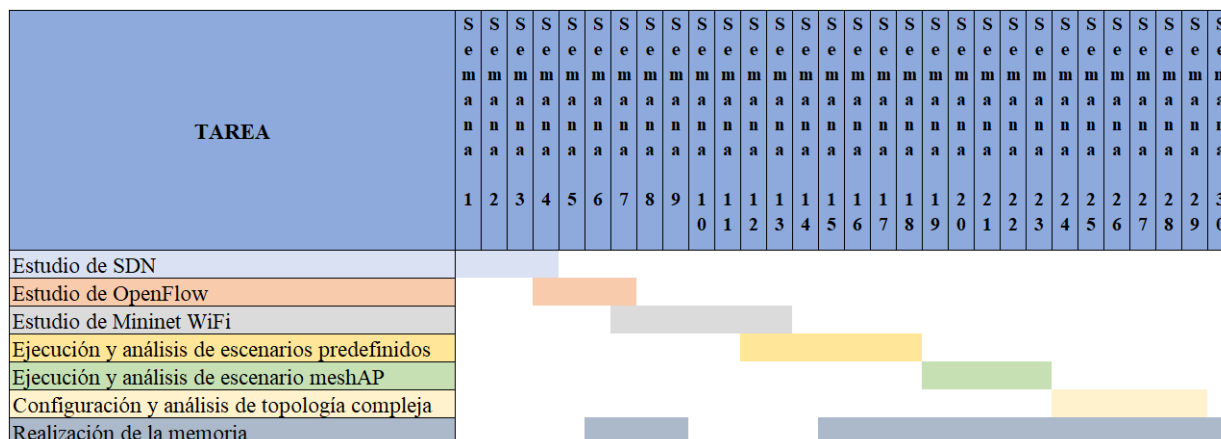


Figura 5.1: Diagrama de Gantt del Trabajo Fin de Grado.

5.3. Líneas futuras

El presente trabajo puede plantearse como punto de partida para futuros proyectos de investigación en el campo de las redes SDN, el protocolo OpenFlow y el emulador Mininet WiFi. Se proponen las siguientes ideas para futuros proyectos:

- Respecto a las redes SDN y WSDN, se podría modificar el script de configuración del Capítulo 4 para incluir nuevas zonas con más nodos y puntos de acceso y examinar si la red reacciona rápidamente a los cambios en la topología y es capaz de encaminar los paquetes sin que las comunicaciones se vean afectadas por un incremento de latencia.
- Dentro del campo del protocolo OpenFlow, la versión empleada por el emulador Mininet WiFi es la 1.0. Este protocolo ya se encuentra actualizado a la versión 1.3, que permite nuevas funcionalidades y tiene un mayor número de campos en su cabecera. Para próximos estudios sería interesante comprobar el funcionamiento de algunas de las topologías de este trabajo en otro emulador que soporte la versión más actualizada del protocolo.
- Crear una topología de red en la que intervengan varios controladores, cada uno en una zona concreta de la misma. En esta red, para establecer comunicación entre zonas con diferente gestión será necesario estudiar el uso de protocolos Este-Oeste, que permiten el intercambio de información y tráfico entre diferentes controladores.

Apéndice A

Script meshAP.py

```
1 #!/usr/bin/python
2
3 '''
4 This example shows on how to create wireless link between two APs
5 with mesh
6 The wireless mesh network is based on IEEE 802.11s
7 '''
8
9 from mininet.log import setLogLevel, info
10 from mn_wifi.link import wmediumd, mesh
11 from mn_wifi.cli import CLI
12 from mn_wifi.net import Mininet_wifi
13 from mn_wifi.wmediumdConnector import interference
14
15
16 def topology():
17     'Create a network.'
18     net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)
19
20     info('*** Creating nodes\n')
21     sta1 = net.addStation('sta1', mac='00:00:00:00:00:11',
22         position='1,1,0')
23     sta2 = net.addStation('sta2', mac='00:00:00:00:00:12',
24         position='31,11,0')
25     ap1 = net.addAccessPoint('ap1', wlans=2, ssid='ssid1',
26         position='10,10,0')
27     ap2 = net.addAccessPoint('ap2', wlans=2, ssid='ssid2',
```



```
28     position='30,10,0')
29 c0 = net.addController('c0')
30
31 info('*** Configuring wifi nodes\n')
32 net.configureWifiNodes()
33
34 info('*** Associating Stations\n')
35 net.addLink(sta1, ap1)
36 net.addLink(sta2, ap2)
37 net.addLink(ap1, intf='ap1-wlan2', cls=mesh, ssid='mesh-ssid',
38     channel=5)
39 net.addLink(ap2, intf='ap2-wlan2', cls=mesh, ssid='mesh-ssid',
40     channel=5)
41
42 info('*** Starting network\n')
43 net.build()
44 c0.start()
45 ap1.start([c0])
46 ap2.start([c0])
47
48 info('*** Running CLI\n')
49 CLI(net)
50
51 info('*** Stopping network\n')
52 net.stop()
53
54
55 if __name__ == '__main__':
56     setLogLevel('info')
57     topology()
```

Apéndice B

Script topo.py

```
1
2 #!/usr/bin/python
3
4 import sys
5
6 from mininet.node import Controller
7 from mininet.log import setLogLevel, info
8 from mn_wifi.link import wmediumd, mesh, ITSLink
9 from mn_wifi.cli import CLI_wifi
10 from mn_wifi.net import Mininet_wifi
11 from mn_wifi.node import OVSBridgeAP
12 from mn_wifi.wmediumdConnector import interference
13
14
15 def topology(stp):
16     "Create a network."
17     net = Mininet_wifi(controller=Controller, link=wmediumd,
18         wmediumd_mode=interference)
19
20     info("*** Creating nodes\n")
21     sta1 = net.addStation('sta1', mac='00:00:00:00:00:01',
22         IP='10.0.0.1/8', position='10,80,0', range=50)
23     sta2 = net.addStation('sta2', mac='00:00:00:00:00:02',
24         IP='10.0.0.2/8', position='20,80,0', range=50)
25     sta3 = net.addStation('sta3', mac='00:00:00:00:00:03',
26         IP='10.0.0.3/8', position='30,80,0', range=50)
27     sta4 = net.addStation('sta4', mac='00:00:00:00:00:04',
```

```

28     IP='10.0.0.4/8', position='100,101,0')
29     sta5 = net.addStation('sta5', mac='00:00:00:00:00:05',
30     IP='10.0.0.5/8', position='50,101,0')
31     sta6 = net.addStation('sta6', mac='00:00:00:00:00:06',
32     IP='10.0.0.6/8', position='50,51,0')
33     sta7 = net.addStation('sta7', mac='00:00:00:00:00:07',
34     IP='10.0.0.7/8', position='100,51,0')
35     sta8 = net.addStation('sta8', mac='00:00:00:00:00:08',
36     IP='10.0.0.8/8', position='40,60,0')
37     sta9 = net.addStation('sta9', mac='00:00:00:00:00:09',
38     IP='10.0.0.9/8', position='35,40,0')
39     sta10 = net.addStation('sta10', mac='00:00:00:00:00:10',
40     IP='10.0.0.10/8', position='30,30,0')
41     sta11 = net.addStation('sta11', mac='00:00:00:00:00:11',
42     IP='10.0.0.11/8', position='40,30,0')
43     if stp:
44         ap4 = net.addAccessPoint('ap4', ssid='new-ssid4',
45         mode='g', channel='1', failMode="standalone",
46         position='100,100,0', stp=True)
47         ap5 = net.addAccessPoint('ap5', ssid='new-ssid5',
48         mode='g', channel='1', failMode="standalone",
49         position='50,100,0', stp=True)
50         ap6 = net.addAccessPoint('ap6', ssid='new-ssid6',
51         mode='g', channel='1', failMode="standalone",
52         position='50,50,0', stp=True)
53         ap7 = net.addAccessPoint('ap7', ssid='new-ssid7',
54         mode='g', channel='1', failMode="standalone",
55         position='100,50,0', stp=True)
56         ap9 = net.addAccessPoint('ap9', wlans=4, ssid='ssid9,,,',
57         mode='g', failMode="standalone",
58         position='30,40,0', stp=True)
59     else:
60         ap4 = net.addAccessPoint('ap4', ssid='new-ssid4',
61         mode='g', channel='1', failMode="standalone",
62         position='100,100,0')
63         ap5 = net.addAccessPoint('ap5', ssid='new-ssid5',
64         mode='g', channel='1', failMode="standalone",
65         position='50,100,0')
66         ap6 = net.addAccessPoint('ap6', ssid='new-ssid6',
67         mode='g', channel='1', failMode="standalone",
68         position='50,50,0')
69         ap7 = net.addAccessPoint('ap7', ssid='new-ssid7',

```

```
mode='g', channel='1', failMode="standalone",
position='100,50,0')

ap9 = net.addAccessPoint('ap9', wlans=4, ssid='ssid9,,,',
mode='g', failMode="standalone", position='30,40,0')
ap10 = net.addAccessPoint('ap10', wlans=2, ssid='ssid10,',
mode='g', failMode="standalone", position='30,35,0')
ap11 = net.addAccessPoint('ap11', wlans=2, ssid='ssid11,',
mode='g', failMode="standalone", position='40,30,0')

ap1 = net.addAccessPoint('ap1', wlans=3, ssid='ssid1,,,',
position='20,100,0', channel='1')
ap2 = net.addAccessPoint('ap2', wlans=3, ssid='ssid2,,,',
position='40,100,0', channel='1')
ap3 = net.addAccessPoint('ap3', wlans=3, ssid='ssid3,,,',
position='40,80,0', channel='1')

c0 = net.addController('c0')

net.setPropagationModel(model="logDistance", exp=5)

info("*** Configuring wifi nodes\n")
net.configureWifiNodes()

info("*** Associating Stations\n")
net.addLink(sta1, ap1)
net.addLink(sta2, ap1)
net.addLink(sta3, ap1)

net.addLink(ap4, sta4)
net.addLink(ap5, sta5)
net.addLink(ap6, sta6)
net.addLink(ap7, sta7)

net.addLink(ap3, sta8)
net.addLink(ap9, sta9)
net.addLink(ap10, sta10)
net.addLink(ap11, sta11)

net.addLink(ap4, ap5)
net.addLink(ap5, ap6)
net.addLink(ap6, ap7)
```

```
112     net.addLink(ap7, ap4)
113     net.addLink(ap2, ap4)
114
115     net.addLink(ap9, ap10)
116     net.addLink(ap9, ap11)
117     net.addLink(ap3, ap9)
118
119     net.addLink(ap1, intf='ap1-wlan3', cls=mesh,
120                 ssid='mesh-ssid', channel='5')
121     net.addLink(ap2, intf='ap2-wlan3', cls=mesh,
122                 ssid='mesh-ssid', channel='5')
123     net.addLink(ap3, intf='ap3-wlan3', cls=mesh,
124                 ssid='mesh-ssid', channel='5')
125
126     net.plotGraph(max_x=150, max_y=150)
127
128     info("*** Starting network\n")
129     net.build()
130     c0.start()
131     ap1.start([c0])
132     ap2.start([c0])
133     ap3.start([c0])
134     ap4.start([c0])
135     ap5.start([c0])
136     ap6.start([c0])
137     ap7.start([c0])
138     ap9.start([c0])
139     ap10.start([c0])
140     ap11.start([c0])
141
142
143     info("*** Running CLI\n")
144     CLI_wifi(net)
145
146     info("*** Stopping network\n")
147     net.stop()
148
149
150 if __name__ == '__main__':
151     setLogLevel('info')
152     stp = True if '-s' in sys.argv else False
153     topology(stp)
```

Bibliografía

- [1] Mireya Tovar Vidal. «Sistemas distribuidos». En: *Facultad de Ciencias de la Computación (BUAP)* (2015).
- [2] Ángel Leonardo Valdivieso Caraguay y col. «SDN: Evolution and Opportunities in the Development IoT Applications». En: *International Journal of Distributed Sensor Networks* (2014).
- [3] I. T. Haque y N. Abu-Ghazaleh. «Wireless Software Defined Networking: A Survey and Taxonomy». En: *IEEE Communications Surveys Tutorials* 18.4 (2016), págs. 2713-2737.
- [4] David Madler. *Introduction to SDN*. 2015. URL: https://www.youtube.com/watch?v=DiChnu_PAzA.
- [5] Open Networking Foundation. «OpenFlow Switch Specification Version 1.3.0». En: *ONF TS-006* (2012).
- [6] Open Networking Foundation. «OpenFlow Switch Specification Version 1.0.0». En: *ONF TS-001* (2009).
- [7] Ramon Fontes y Christian Rothenberg. *Wireless Network Emulation with Mininet-WiFi*. 2019.
- [8] Ramon Fontes y Christian Rothenberg. *Mininet-WiFi: the user manual*. 2019.
- [9] Airberry. *Introduction to Mesh Networks*. 2012.