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

Índice general

Índice de figuras

Índice de tablas

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

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

WSDN Wireless Software Defined Networking

LAN Local Area Network

WAN Wide Area Network

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 NFV.

En este sentido,

1.2. Objetivos

1.3. Estructura de la memoria

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. Estos protocolos permiten construir una comunicación transparente entre los componentes heterogéneos que forman la red, mediante la interconexión de las diferentes capas de estos protocolos. Cuentan con grandes ventajas que avalan su uso actualmente, algunas de ellas se exponen a continuación brevemente: [1]

- Economizan los recursos disponibles, son más veloces y fiables.
- Facilitan la comunicación entre dispositivos y personas.
- Reparten la carga de tráfico entre las distintas máquinas que componen la red de forma eficaz y flexible.

Sin embargo, también existen una serie de desventajas en ellos y en la red resultante que han motivado la investigación y el desarrollo de otras alternativas que hagan la red aún más inteligente y eficiente.

- No existe mucho software que permita la optimización de los sistemas distribuidos actualmente.

- 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 puede ser solventados mediante recursos hardware.
- Aunque la red es segura, hay puntos en los que convendría mejorar los sistemas de seguridad y confidencialidad.

Estos sistemas empleados en la red tradicional no fueron diseñados para soportar una escalabilidad, tráfico y mobilidad crecientes de la incipiente 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, 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), sobre el que se encuentra un sistema operativo, que es habitualmente Linux, que recibe información que le envía el hardware y ejecuta la aplicación del plano de control. Esta aplicación es básicamente un software con numerosas líneas de código, que sirven para identificar los saltos que deben seguir los paquetes para llegar a su destino final. Este programa es específico y sigue los estándares que el proveedor de servicios configura en él, los más modernos llegando incluso a detectar y descartar paquetes maliciosos e intrusiones ilegales mediante *firewalls* e IDS. [2] (pág. 2)

Sin embargo, los operadores de red encuentran una limitación a la hora de administrar su red: la red resultante es muy rígida y solo tienen la capacidad de configurar algunos de sus parámetros, encontrándose a merced de la configuración realizada por el proveedor de servicios. Si el administrador u operador de red quisiera modificar la ruta de un paquete, tendría que estudiar los parámetros de la red y la prioridad de las reglas para poder hacerlo, porque con este modelo no se tiene una imagen global de la red. [2] (pág. 2)

Además, la división entre software y hardware en las redes actuales es latente y proporciona la ventaja de poder actualizar el comportamiento de la red simplemente actualizando el software de la aplicación. Sin embargo, muchas veces este cambio de software depende del hardware o es necesario que sea desarrollado por el proveedor de los servicios, llegando incluso a pasar varios años hasta que se consigue. 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*. 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. [2] (pág. 2)

2.1.2. SDN: concepto y arquitectura

WSDN es una evolución de SDN, en la 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. El *Software Defined Networking* 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 un plano de control más centralizado, en el 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)

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)

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 control, 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 control que recolectan información de la misma para decidir de manera coordinada la configuración de cada uno de los recursos y elementos de la red. 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

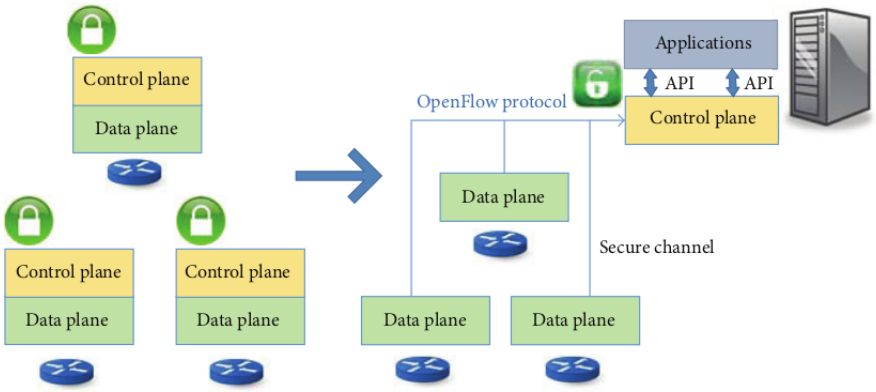

Figura 2.1: Comparación de arquitectura de red tradicional y SDN.

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 de la red. [3] (pág. 1)

2.1.3. NOS de la red o controlador

Las redes SDN se diferencian de las redes tradicionales en que existe la figura centralizada del controlador. También llamado NOS (por sus siglas en ingles, *Network Operating System*), 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 NOS se comunica, según el criterio de los expertos, con dos interfaces: la interfaz de bajo nivel *southbound interface* y la interfaz de alto nivel *northbound interface*. Se conoce como *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 ellas se encuentra el software puro del controlador, que mediante dos APIs se comunica con estas interfaces. En la Figura 2.2 muestra cómo es la estructura del NOS 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 (*southbound interface*), el switch consulta su tabla de flujos para encaminar al paquete hacia su destino. 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. Son las

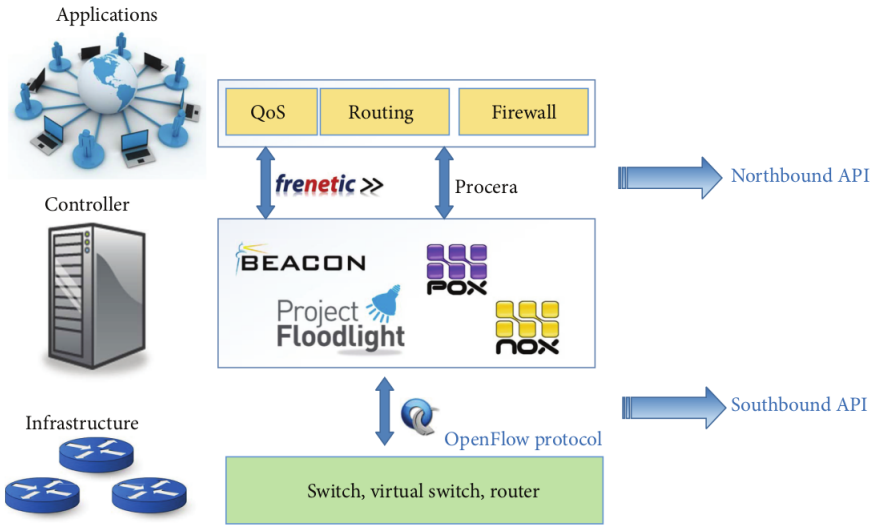

Figura 2.2: Estructura del NOS.

apelaciones de red (*northbound interface*) las que deciden qué hacer con el paquete a los servicios. En esa estructura, el controlador que se encuentra entre las interfaces hace de traductor para permitir la comunicación entre ellas. [4]

Southbound interface y *forwarding devices*

Los servicios de nivel bajo pueden ser switches hardware o software. Los switches hardware usan el protocolo OpenFLow y aportan mayor funcionalidad a la red. Los switches software emplean OVS, no aportan tanta versatilidad a las funciones de red, pero es más flexible a los cambios de comportamiento de la red. Ambos tipos de switches se comunican con el controlador mediante la interfaz de bajo nivel usando OpenFlow o OVS. 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]

Network Operating System

El NOS o controlador puro de la red está formado por:[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.
- *Host tracking*: servicio para descubrir las direcciones IP y MAC de los elementos de la red.

Northbound interface y *network applications*

Las aplicaciones de red 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 cual llega al controlador información del core de la red y órdenes de control del comportamiento de la red. Existe otro tipo de interfaz entre controlador y aplicaciones de red: las APIs Java, interfaces programadas que también permiten el intercambio de información. [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 cae una interfaz o sistema, la propia red se encarga de reencaminar el tráfico por otros sistemas e interfaces que se encuentren en funcionamiento, reduciendo así tiempos de indisponibilidades y errores. Por otro lado, la red SDN es más escalable que la tradicional porque es capaz de manejar un mayor número de solicitudes al realizarse todas las operaciones mediante uso de software. 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*. [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 distribuidos, como BGP, MPLS o OSPF, que suelen dar errores de configuración. En este plano es en el que se forman y albergan las políticas de red, decide cómo se tratan los paquetes y envían esta información al plano de datos. Esto provoca que en la red tradicional no se pueda intervenir directamente en el plano de datos para modificar el comportamiento de la red, sino que hay que hacerlo sobre las políticas definidas en el plano de control para poder cambiar el manejo de tráfico. [4]

Para acceder al plano de control en la red tradicional hay que hacerlo mediante órdenes y comandos, por lo que también existen limitaciones en la intervención sobre el plano de datos. 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, sino que debe ser general y para poder soportar las tablas de flujos y configuraciones instaladas por el controlador en él. [4]

2.2. OpenFlow

Surgió como un 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. En la misma línea que SDN, OpenFlow también emplea un controlador y un switch, además de un canal seguro de comunicación entre los dos. [2] (pág. 3)

La principal ventaja con la que cuenta OpenFlow es que emplea elementos hardware que suelen estar disponibles en los equipos. Igual que otros protocolos empleados, usa tablas de encaminamiento en las que conviven órdenes o reglas para los diferentes paquetes o tipos de tráfico. De este modo, cuando un paquete llega al equipo, se lee su cabecera, se consulta la entrada en la tabla de encaminamiento 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. [2] (pág. 3)

La versión que emplea Mininet WiFi es OpenFlow 1.0.0. Sus principales mejoras respecto a las versiones anteriores son las siguientes: [5] (págs. 96, 97)

- Incluye colas en los puertos, de este modo se consigue reducir el ancho de banda necesario. Cuando hay muchos paquetes que llegan a un puerto para ser enviados, es frecuente que algunos de ellos sean descartados por el switch debido a que su velocidad de procesamiento y envío no es lo suficientemente alta, o incluso porque la velocidad del medio de

transmisión no lo permite. En estos casos es muy útil el buffer que se incluye con la versión 1.0.0 de OpenFlow, pues permite almacenar los paquetes en las colas de los puertos hasta que son procesados y enviados, reduciendo la tasa de descarte de paquetes.

- También se introduce en esta versión del protocolo un identificador de flujos que funciona como una cookie. Cuando un flujo nuevo es instalado en la tabla de flujos del switch o del access point, se guarda también en la entrada de la tabla de flujos esta cookie que le envía el controlador. Este identificador en forma de cookie se emplea en el reporte de estadísticas que necesita el controlador para conocer el comportamiento de la red.
- Nuevo campo con una descripción del switch indicada por el propietario.
- En esta versión también se incluyen respecto a las anteriores las direcciones IP origen y destino del tráfico descrito en el flujo de datos para paquetes ARP, de modo que sea más sencillo encontrar la entrada de la tabla de flujos que mejor encaje con el paquete recibido.
- También incluye un campo nuevo para poder comparar con el tipo de servicio IP del tráfico.
- Permite la extracción de estadísticas de un único puerto mediante el campo *port_no*, donde se indica el número de puerto del que se quieren obtener estadísticas. La opción *OFPP_NONE* permite extraerlas de todos los puertos.
- Se indica la validez de las estadísticas y de los mensajes en nanosegundos, en lugar de en milisegundos como se indicaba en versiones más antiguas.

A continuación se exponen los elementos y características de OpenFlow, entre ellos se encuentran los switches de la red, los puertos, los campos de los paquetes OpenFlow, los tipos de mensajes enviados y las estadísticas que mejoran su funcionamiento.

2.2.1. Puertos OpenFlow

Se conoce como puerto a la interfaz por la que se reciben o envían paquetes en una máquina. En OpenFlow los switches se conectan de manera lógica con otros mediante puertos OpenFlow, de esta manera cada switch tendrá asignados una serie de puertos para este protocolo, pudiéndose quedar puertos del switch sin asignar a él si no fueran necesarios. 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 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 función de cual sea su origen. [5] (pág. 8)

Igual que en las redes tradicionales, en OpenFlow existen diferentes tipos de puertos. En concreto en la versión 1.0.0 se usan los puertos físicos y los puertos virtuales, y en la versión 1.3 aparecen los puertos reservados. [5] (pág. 8)

Puertos físicos

Se llama puertos físicos a las interfaces hardware de los routers o switches. Existe correspondencia directa entre ambos, de manera que en una máquina es imposible que haya un número diferente de puertos físicos y de interfaces. Hay ocasiones en las que se rompe esta regla: cuando se virtualiza un switch OpenFlow sobre hardware, en cuyo caso la correspondencia será entre puerto "físico.^{ei}nterfaz virtual. [5] (pág. 9)

Puertos lógicos

En este caso cada puerto lógico no corresponde con una interfaz hardware del switch necesariamente, puede haber varios puertos físicos en una misma interfaz o no haber ninguno, y usualmente un mismo puerto lógico pasa por varios puertos físicos. Generalmente, el tráfico que transita por un puerto lógico se encapsula. Generalmente, la asociación entre puerto lógico y los puertos físicos se materializa en las técnicas de tunneling. El tunneling se emplea para crear túneles cuyo origen y destino son máquinas pertenecientes a la red; hay un puerto lógico asignado a cada túnel que se asocia también a cada una de las interfaces o puertos físicos de los switches o máquinas que se encuentran entre el origen y el destino. En los paquetes de tráfico se indica a qué túnel pertenece cada uno de ellos mediante el campo *Tunnel-ID*. En la versión 1.0.0 se emplean puertos virtuales para el envío de paquete por inundación o para indicar cuál es el puerto de entrada al switch. [6] (pág. 9)

2.2.2. 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 los switches 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 de 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. En el epígrafe 2.2.3 se expone con mayor profundidad la diferencia entre estos dos tipos de switches. [5] (pág. 10)

2.2.3. Tablas de flujos

OpenFLow basa sus operaciones en la llamada tabla de flujos. 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 la 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 de la cabecera de los mensajes OpenFlow se emplean para consultar las tablas de flujos y encontrar la entrada que coincide con el tráfico recibido. Para que un paquete se considere que cumple con los requisitos de una entrada ha de encajar en todos los campos de dicha entrada. No es necesario que los 12 campos estén completos en todas las entradas, con lo cual puede suceder que un mismo paquete encaje con dos entradas con acciones diferentes. Para evitar estas colisiones 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. Además, para hacer aún más exhaustiva la búsqueda hay switches OpenFlow que permiten comparar las direcciones IP con máscara de subred. 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)

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.

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 podría decirse que 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 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)

Las entradas de las tablas de flujos están asociadas con acciones que debe ejecutar el switch sobre el paquete recibido. Las entradas 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 ejecutar las acciones 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 a esa información el controlador establecerá las acciones pertinentes para que no haya errores durante el procesado de paquetes. En el apartado 2.2.2 se indica que existen dos tipos de switches OpenFlow y dependiendo de dicho tipo se soportan unas acciones u otras. [6] (págs. 3, 6)

Las acciones soportadas por los dos tipos de 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 árbol de inundación mínimo 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.

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 *checksum*.
Transport destination port	16	Actualiza el puerto destino del nivel de transporte (TCP o UDP) y modifica el valor del *checksum*.

Tabla 2.2: Acciones opcionales de modificación de campos.

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 gestión eficiente de los recursos disponibles y del comportamiento que presenta la red, 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)

2.2.4. 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. No obstante, estos mensajes del switch tienen que seguir el formato del protocolo OpenFlow para que sea entendido por el controlador. [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 y 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-inçuando 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 procesado 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.

Tras esta visión detallada del protocolo OpenFlow v1.0.0 se esconde el funcionamiento real del protocolo. En el capítulo 3 se expondrá de manera práctica y apoyándose en un escenario configurado 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.

2.2.5. Arquitectura en OpenFLow

Tras todos los detalles indicados en los epígrafes anteriores, en la Figura 2.3 se muestra la arquitectura de OpenFlow con la excusa de ofrecer una vista general de los conceptos anteriormente explicados. [2] (pág. 4)

2.3. Mininet WiFi

Mininet WiFi es un emulador de redes SDN inalámbricas. Es la evolución del emulador de redes SDN Mininet. Los emuladores son empleados por los investigadores para probar el

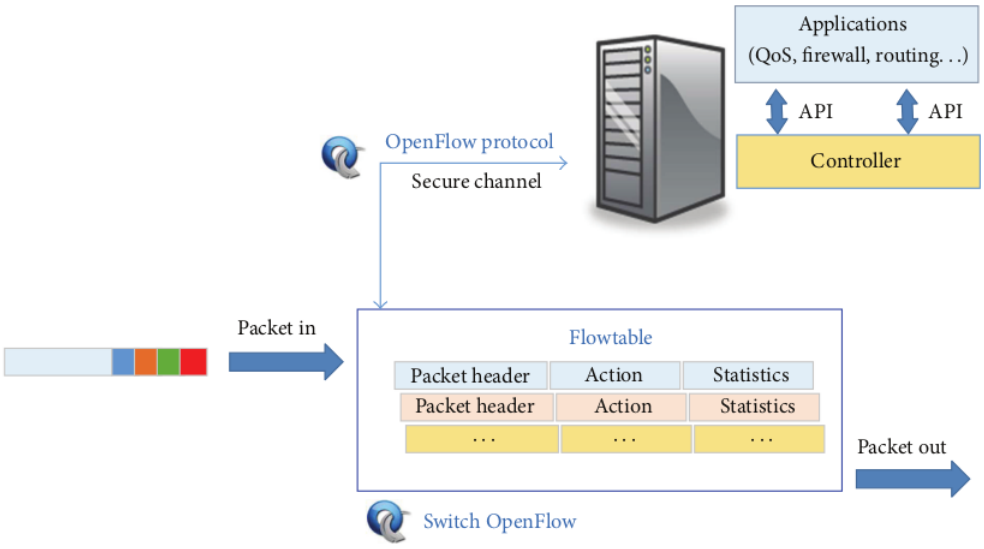

Figura 2.3: Arquitectura de OpenFlow.

comportamiento de una red antes de su despliegue. Ofrecen una interfaz que permite realizar experimentos con la premisa de obtener resultados basados en el realismo, pero sin olvidar 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 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 dento 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 los *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

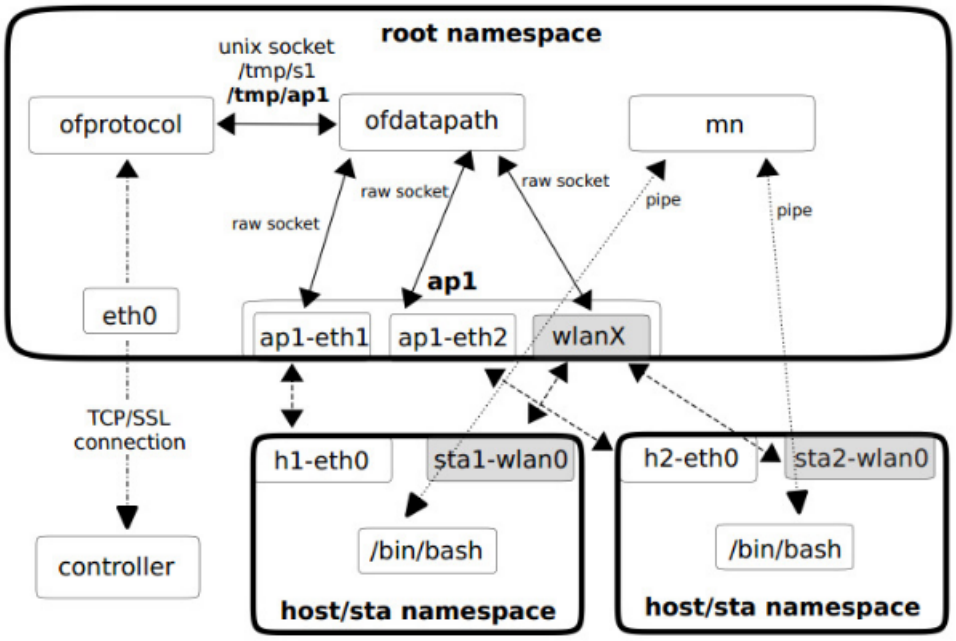

Figura 2.4: Arquitectura de Mininet WiFi.

máquinas físicas reales con sus propias rutas, firewalls y servicios de red. En la Figura 2.4 se muestra la arquitectura de Mininet WiFi sobre los *linux namespaces*. [7] (pág. 15)

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

- Controlador
- *Access Points*
- *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 segunda opción, para mantener el entorno actualizado a las diferentes versiones que se vayan subiendo al sitio inicial 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. A posteriori a lo largo del desarrollo del trabajo se explica en mayor profundidad cómo funciona el intercambio de paquetes usando el protocolo OpenFlow y cómo el controlador 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

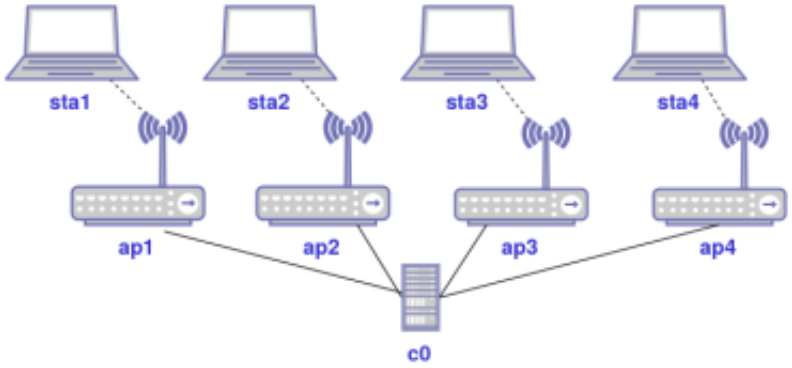

Figura 2.5: Topología lineal con 4 *hosts*.

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 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.

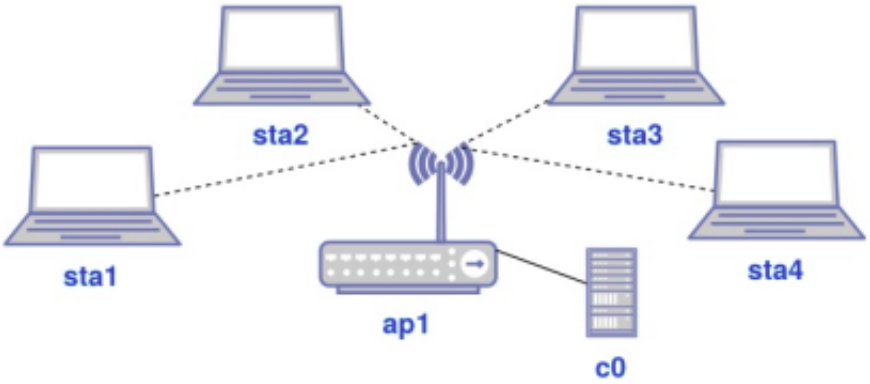

Figura 2.6: Topología sencilla con 4 *hosts*.

Parámetro	**Valores**
–switch=SWITCH	default | ivs | lxbr | ovs | ovsbr | ovsk | user[,param=value...] ovs=OVSSwitch default=OVSSwitch ovsk=OVSSwitch lxbr=LinuxBridge user=UserSwitch ivs=IVSSwitch ovsbr=OVSBridge
–ap=AP	default | ivs | lxbr | ovs | ovsbr | ovsk | user[,param=value...] ovs=OVSAP default=OVSAP ovsk=OVSAP lxbr=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.

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

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 porque este tipo de redes 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.

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

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

Por último, se incluirían estos comandos:

```
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. 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.

Para ejecutarlo habría que poner en la terminal la siguiente línea:

`sudo python` *`nombre_script`*`.py`

Según las funciones empleadas, al comienzo del script, tras una primera línea `#!/ usr/bin/python` se deben importar las mismas de cada uno de los módulos a los que pertenecen. En concreto en este caso se escribirían 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
```

A continuación nos saldrán varias líneas indicando que la red se está creando y configurando. Una vez termine de configurar ya se podrán escribir en el *prompt* 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 va viendo su funcionamiento.

Capítulo 3

Análisis script meshAP.py

3.1. Configuración del script meshAP.py

Este fichero 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. El contenido del script meshAP.py se encuentra disponible para su consulta en el A.

Se trata de una red mesh de medio inalámbrico (`link=wmedium`) que presenta interferencias (`wmedium mode=interference`). El simulador se encarga de calcular el nivel de interferencia en base a la distancia existente entre un nodo y sus nodos adyacentes. También se desprende del script que la red presenta el controlador propio de las redes definidas por software, conocidas como redes SDN por sus siglas en inglés.

Los dos APs de la red mesh configurada en el script presentan cada uno de ellos dos WLANs, 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.

3.2. Topología inicial

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.

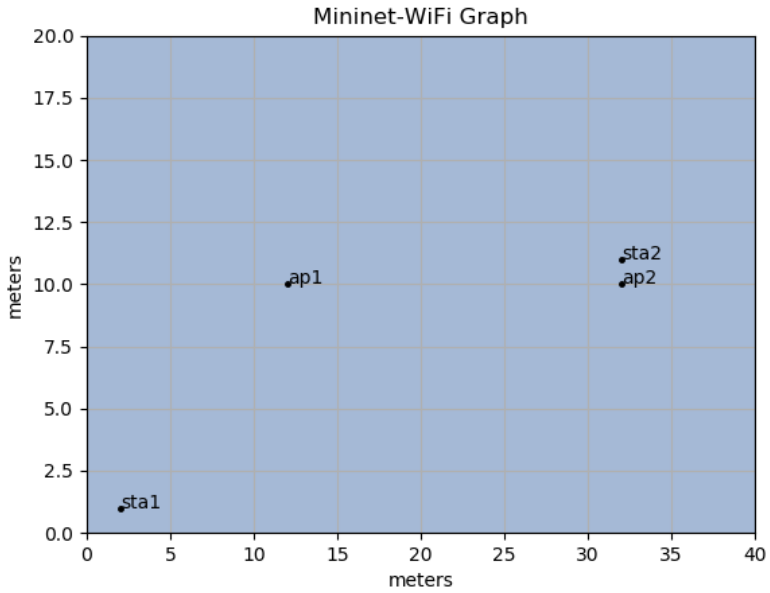

Figura 3.1: Posición de los nodos de la red meshAP.

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.

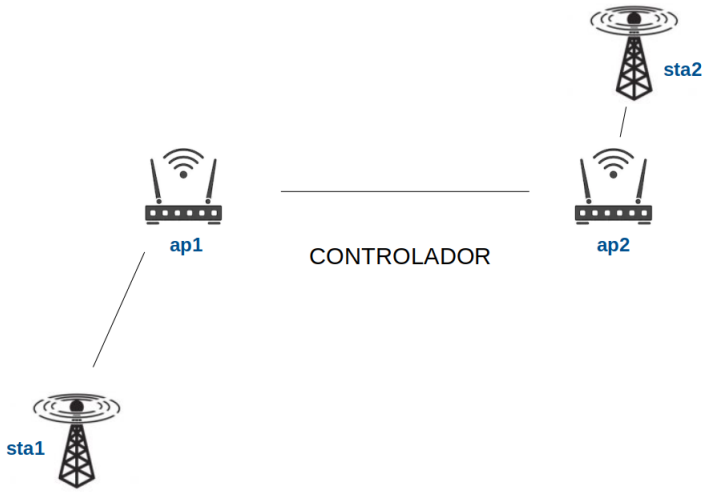

Figura 3.2: Topología de la red meshAP.

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 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.

3.3. Establecimiento de la conexión y configuración de las interfaces

3.4. Análisis de los flujos de datos con `pingall`

Hay que recordar que el comando `pingall` se encarga de ejecutar un mensaje *ping* entre todas las estaciones base que componen una red. En la red meshAP, 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 `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 de broadcast (ff:ff:ff:ff:ff:ff). En este mensaje, origen pregunta quién tiene 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 3.5 del presente capítulo.

A continuación se analiza el contenido de las tablas de flujos del controlador `c0` de la red, y de los puntos de acceso `ap1` y `ap2`. Como se ha indicado, estas tablas deberían tener entradas para todos los tipos de tráfico que se envían entre los nodos, tras ejecutar el comando `pingall`.

El *datapath* del controlador de la red tiene los siguientes puertos e interfaces, 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)
```

Para comprobar los flujos que se instalan en el controlador, se debe ejecutar el siguiente comando:

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

Regla 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
```

Regla 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
```

Regla 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
```

Regla 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
```

Regla 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
```

Regla 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,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'
```

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,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:'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,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:'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 di-

rección 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,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:'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,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:'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,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:'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'
```

3.5. Capturando el tráfico OpenFlow entre el controlador y los *access points*

Para analizar el intercambio de paquetes capturados cuando se ejecuta el comando `pingall` se necesita conocer los puertos configurados en los *access points* `ap1` y `ap2`. 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
```

Tras capturar los paquetes en la interfaz `loopback:lo`, se realiza el análisis que sigue.

El primer `ping` que se realiza es desde `sta1` hacia `sta2`, por lo que `ap1` es el primer punto de acceso que 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.

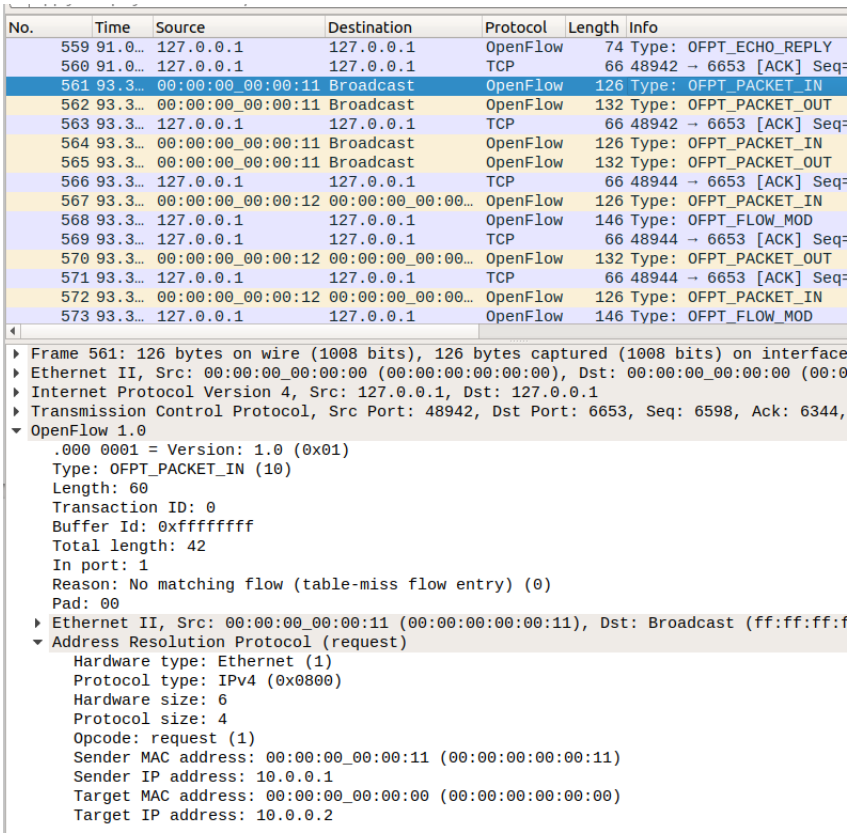

Figura 3.3: Consulta de `ap1` al controlador

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. En la Figura 3.4 se muestra el paquete capturado.

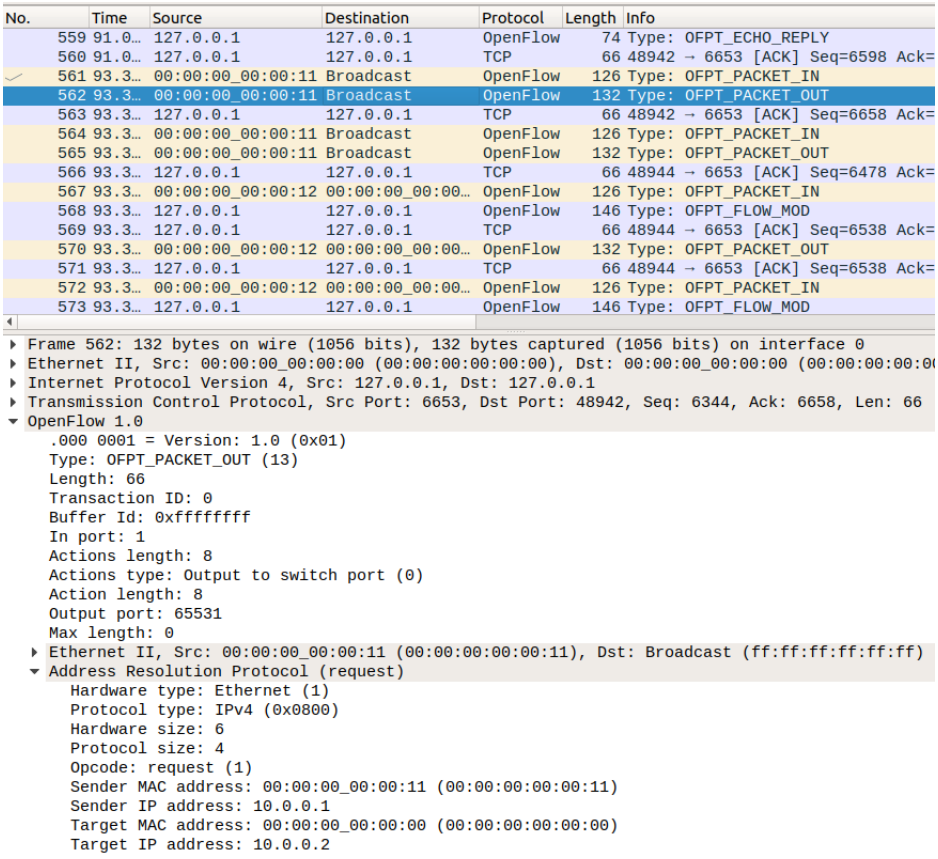

Figura 3.4: Respuesta del controlador a ap1

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 expulsar el paquete pos el resto de sus puertos.

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

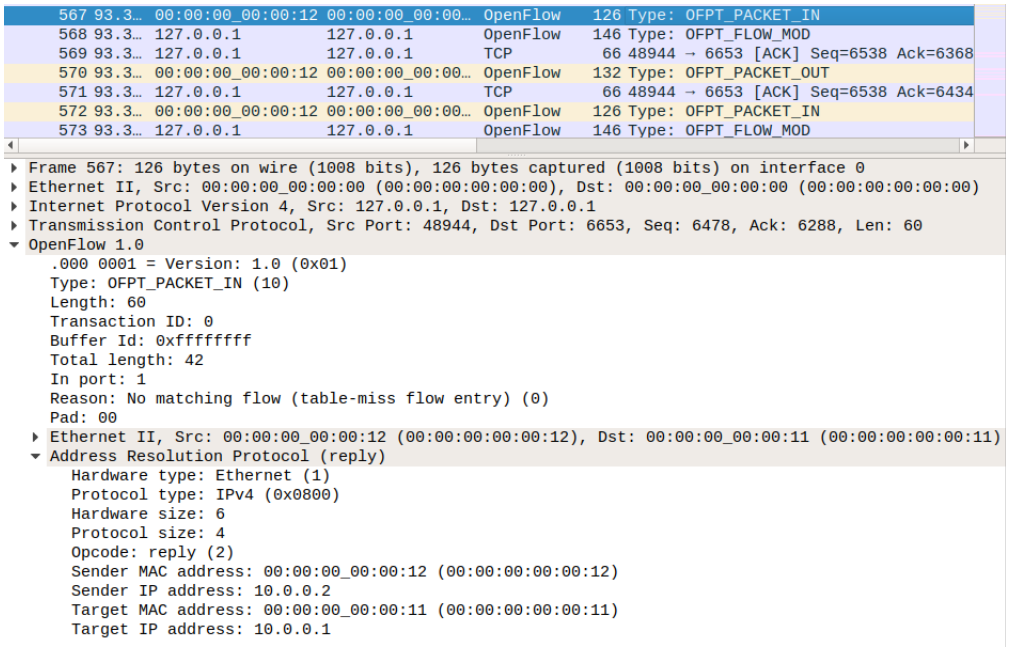

Figura 3.5: Consulta de ap2 al controlador

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 meshAP/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 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 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 3.4 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:
```

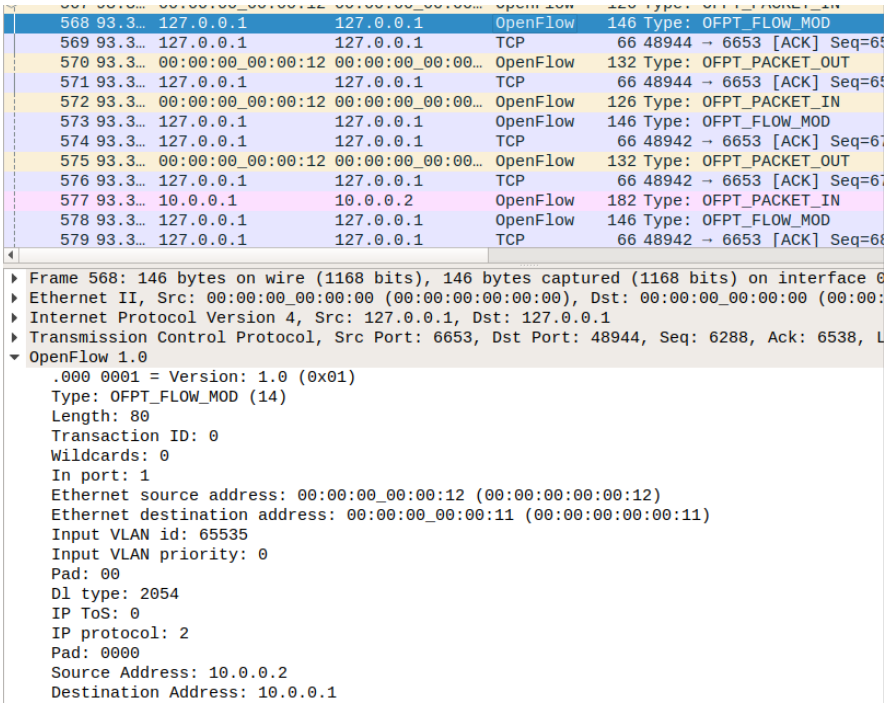

Figura 3.6: Respuesta de modificación del controlador a ap2

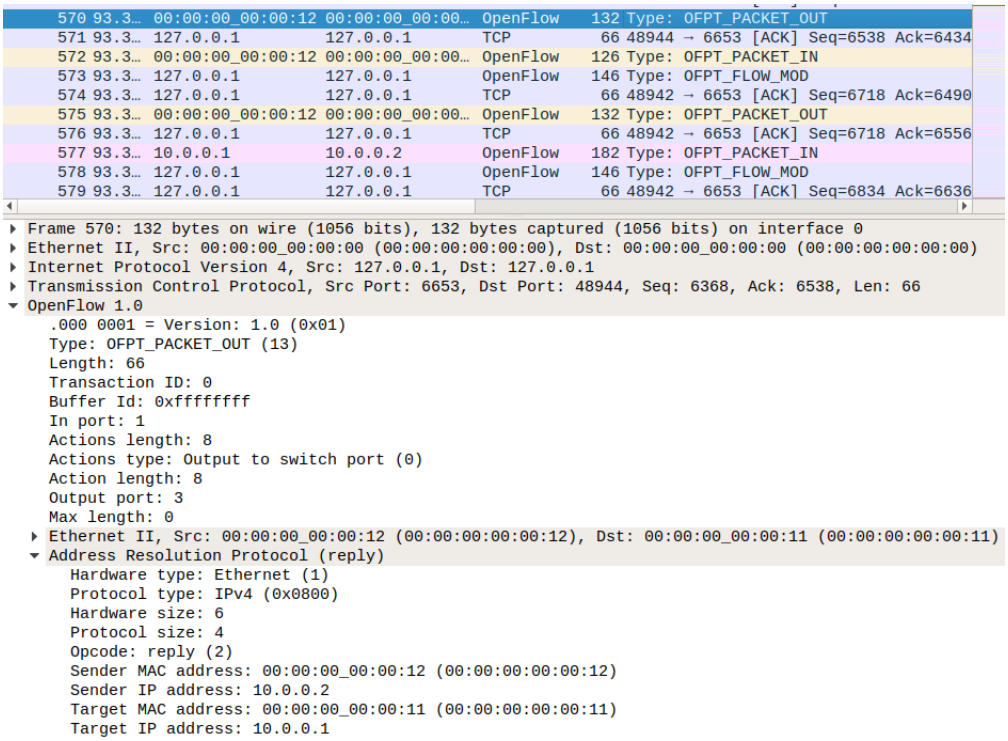

Figura 3.7: Respuesta del controlador con las acciones a instalar en ap2

```
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:
```

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.

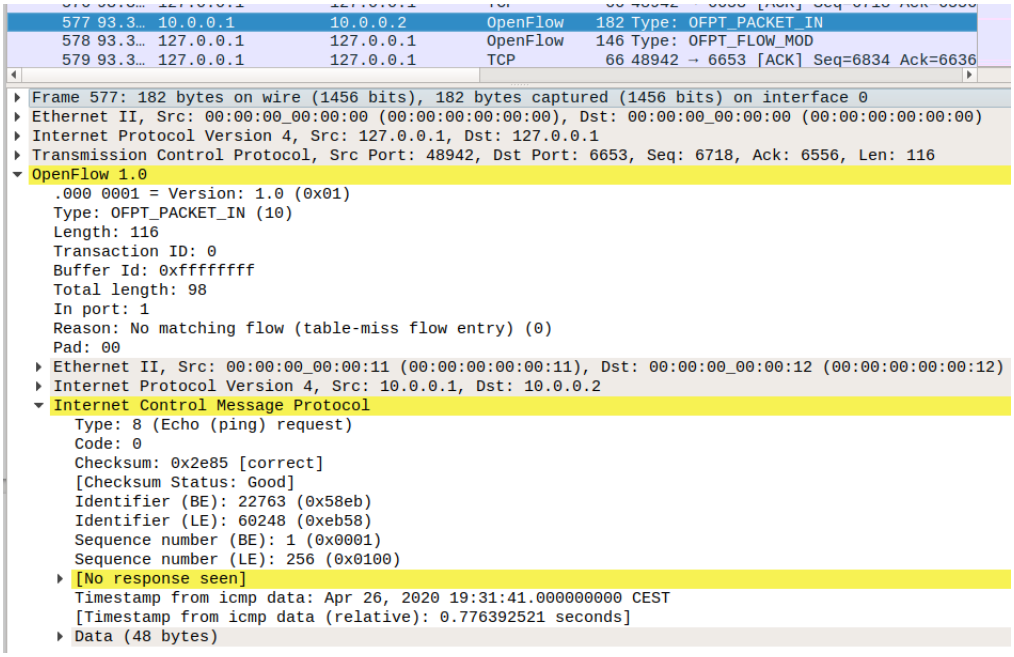

Figura 3.8: Consulta de `ap1` del `ICMP echo request`

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 meshAP/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-wlan1' 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-wlan1'.

Con esta información de las conexiones entre los nodos es sencillo completar el esquema de la topología de la red mostrada en en el apartado 3.2. La topología resultante se muestra en el apartado 3.6.

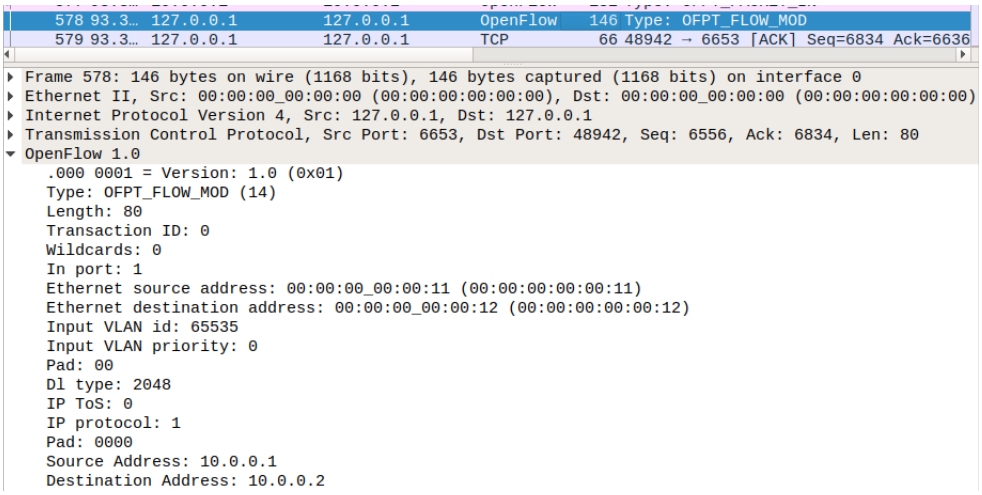

Figura 3.9: Respuesta del controlador con la modificación del flujo en ap2

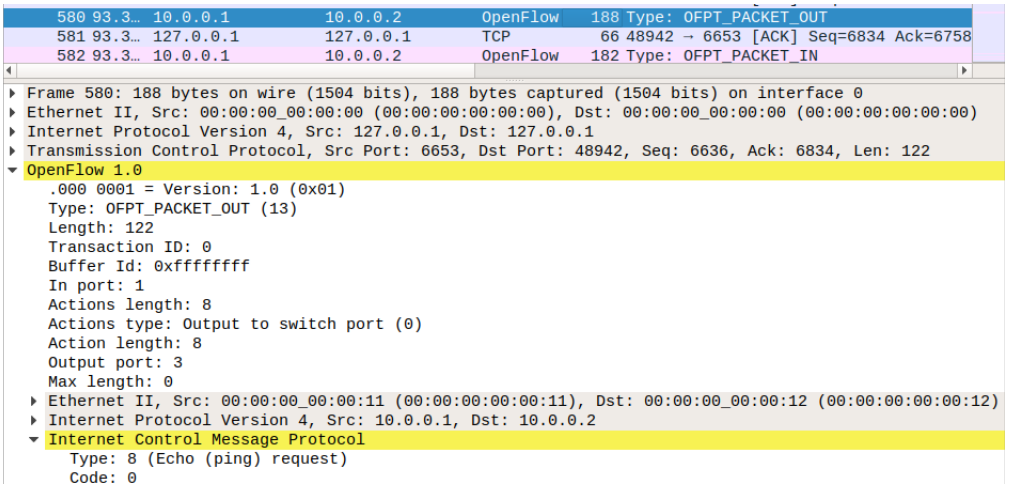

Figura 3.10: Respuesta del controlador con las acciones a instalar en ap1

3.6. 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 3.5, 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 estaciones base que se comunican entre ellas. Además de por motivos físicos, también respalda la unicidad de los *access points* las propiedades

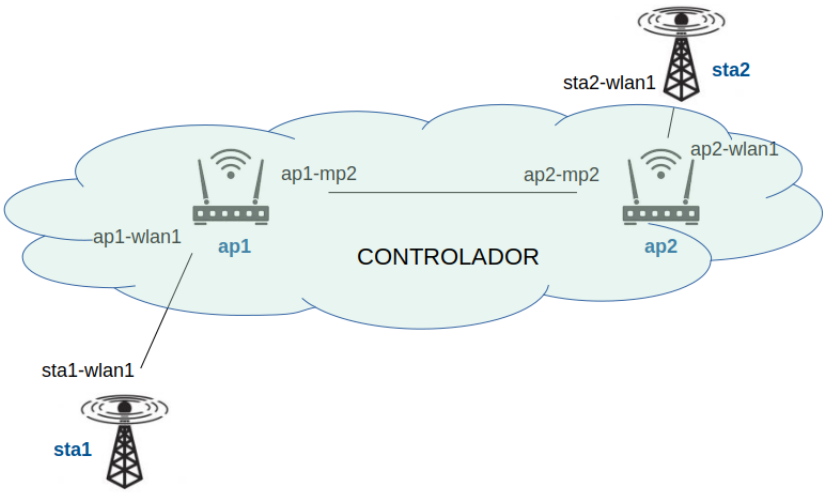

Figura 3.11: Topología resultante de la red meshAP.

características de las redes SDN: 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 autoequilibre. 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 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 exponencialmente, 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

La configuración realizada se encuentra en el Apéndice B.

La topología de la red diseñada es la representada en la Figura 4.1. La red ideada 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, 9 APs, 14 enlaces inalámbricos y 7 enlaces cableados.

De la topología configurada y mostrada en la Figura 4.1 las 7 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 y ap9 con ap10. 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 que el receptor reciba toda la amplitud de la señal enviada por el equipo transmisor).

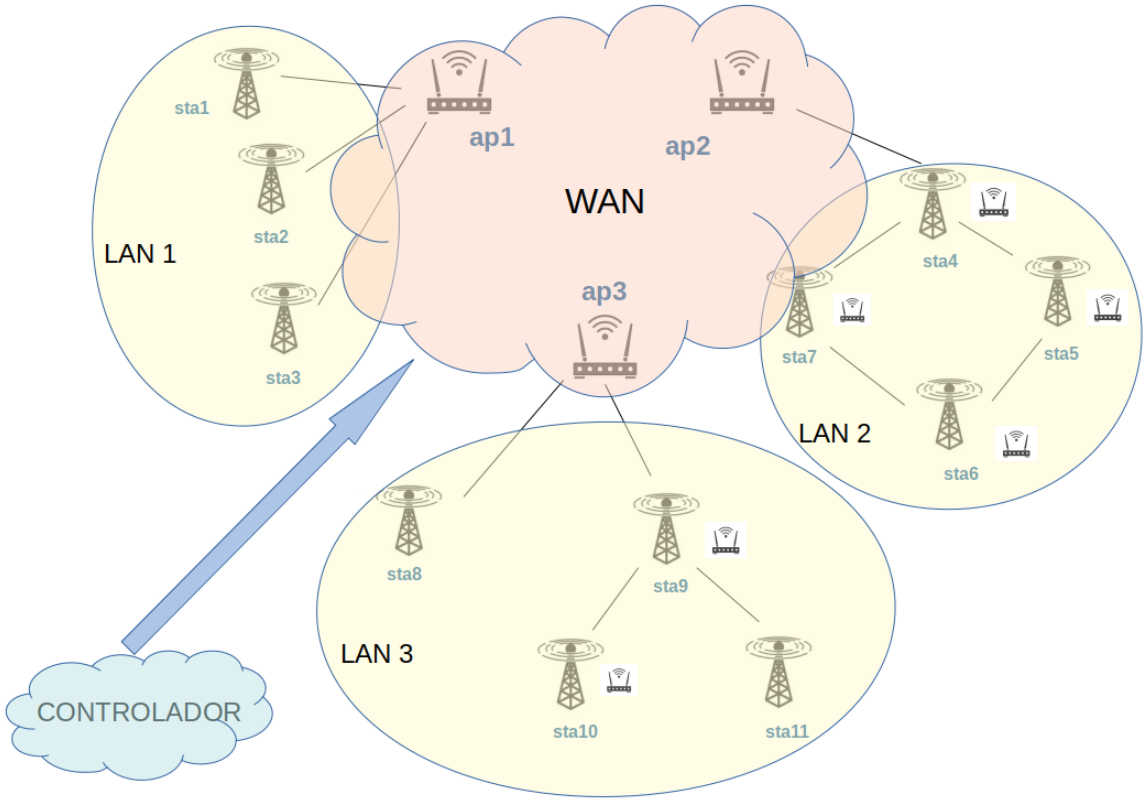

Figura 4.1: Topología de la red personalizada

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

4.2.1. Conectividad y lantencia

La primera prueba que se hizo con estos elementos fue comprobar que hubiera conexión real entre todas las estaciones base. Para ello se empleó el comando `pingall`, que arrojó que todas ellas tenían comunicación. 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 4.1 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 4.1: Latencias en milisegundos (ms) obtenidas tras el test de conectividad de la red.

La Tabla 4.1 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 las cachés que almacenan las direcciones MAC de las máquinas con las que se comunica el equipo, así como 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 tanto en las cachés de los equipos de red y 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 en determinados equipos puede haber tablas infinitas con entradas en desuso o erróneas, y que choca con los principios de versatilidad y autogestión de las redes

SDN, ya que supondría una red estática en la que los nodos no sufren caídas ni alteraciones de sus rutas.

4.2.2. 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 4.2.

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

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 la *performance* de ancho de banda disponible que presentan. Puede 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.

4.3. Análisis de las diferentes áreas de la red

4.3.1. WAN

4.3.2. LAN 1

4.3.3. LAN 2

4.3.4. LAN 3

Apéndice A

Script meshAP.py

```
#!/usr/bin/python

'''
This example shows on how to create wireless link between two APs
with mesh
The wireless mesh network is based on IEEE 802.11s
'''

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


def topology():
'Create a network.'
net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)

info('*** Creating nodes\n')
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',
```

```
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, f=5)
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, f=5)
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, f=5)
27     sta4 = net.addStation('sta4', mac='00:00:00:00:00:04',
```

```
28          IP='10.0.0.4/8', position='100,101,0', f=5)
29      sta5 = net.addStation('sta5', mac='00:00:00:00:00:05',
30          IP='10.0.0.5/8', position='50,101,0', f=5)
31      sta6 = net.addStation('sta6', mac='00:00:00:00:00:06',
32          IP='10.0.0.6/8', position='50,51,0', f=5)
33      sta7 = net.addStation('sta7', mac='00:00:00:00:00:07',
34          IP='10.0.0.7/8', position='100,51,0', f=5)
35      sta8 = net.addStation('sta8', mac='00:00:00:00:00:08',
36          IP='10.0.0.8/8', position='40,60,0', f=5)
37      sta9 = net.addStation('sta9', mac='00:00:00:00:00:09',
38          IP='10.0.0.9/8', position='35,40,0', f=5)
39      sta10 = net.addStation('sta10', mac='00:00:00:00:00:10',
40          IP='10.0.0.10/8', position='30,30,0', f=5)
41      sta11 = net.addStation('sta11', mac='00:00:00:00:00:11',
42          IP='10.0.0.11/8', position='40,30,0', f=5)
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          ap10 = net.addAccessPoint('ap10', wlans=2, ssid='ssid10,',
60              mode='g', failMode="standalone",
61              position='30,35,0', stp=True)
62      else:
63          ap4 = net.addAccessPoint('ap4', ssid='new-ssid4',
64              mode='g', channel='1',failMode="standalone",
65              position='100,100,0')
66          ap5 = net.addAccessPoint('ap5', ssid='new-ssid5',
67              mode='g', channel='1', failMode="standalone",
68              position='50,100,0')
69          ap6 = net.addAccessPoint('ap6', ssid='new-ssid6',
```

```
70              mode='g', channel='1', failMode="standalone",
71              position='50,50,0')
72          ap7 = net.addAccessPoint('ap7', ssid='new-ssid7',
73              mode='g', channel='1', failMode="standalone",
74              position='100,50,0')
75          ap9 = net.addAccessPoint('ap9', wlans=4, ssid='ssid9,,,',
76              mode='g', failMode="standalone", position='30,40,0')
77          ap10 = net.addAccessPoint('ap10', wlans=2, ssid='ssid10,',
78              mode='g', failMode="standalone", position='30,35,0')
79
80      ap1 = net.addAccessPoint('ap1', wlans=3, ssid='ssid1,,',
81          position='20,100,0', channel='1')
82      ap2 = net.addAccessPoint('ap2', wlans=3, ssid='ssid2,,',
83          position='40,100,0', channel='1')
84      ap3 = net.addAccessPoint('ap3', wlans=3, ssid='ssid3,,',
85          position='40,80,0', channel='1')
86
87      c0 = net.addController('c0')
88
89      net.setPropagationModel(model="logDistance", exp=5)
90
91      info("*** Configuring wifi nodes\n")
92      net.configureWifiNodes()
93
94      info("*** Associating Stations\n")
95      net.addLink(sta1, ap1)
96      net.addLink(sta2, ap1)
97      net.addLink(sta3, ap1)
98
99      net.addLink(ap4, sta4)
100     net.addLink(ap5, sta5)
101     net.addLink(ap6, sta6)
102     net.addLink(ap7, sta7)
103
104     net.addLink(ap3, sta8)
105     net.addLink(ap9, sta9)
106     net.addLink(ap10, sta10)
107     net.addLink(ap9, sta11)
108
109     net.addLink(ap4, ap5)
110     net.addLink(ap5, ap6)
111     net.addLink(ap6, ap7)
```

```
    net.addLink(ap7, ap4)
    net.addLink(ap2, ap4)

    net.addLink(ap9, ap10)
    net.addLink(ap3, ap9)

    net.addLink(ap1, intf='ap1-wlan3', cls=mesh,
        ssid='mesh-ssid', channel='5')
    net.addLink(ap2, intf='ap2-wlan3', cls=mesh,
        ssid='mesh-ssid', channel='5')
    net.addLink(ap3, intf='ap3-wlan3', cls=mesh,
        ssid='mesh-ssid', channel='5')

    net.plotGraph(max_x=150, max_y=150)

    info("*** Starting network\n")
    net.build()
    c0.start()
    ap1.start([c0])
    ap2.start([c0])
    ap3.start([c0])
    ap4.start([c0])
    ap5.start([c0])
    ap6.start([c0])
    ap7.start([c0])
    ap9.start([c0])
    ap10.start([c0])


    info("*** Running CLI\n")
    CLI_wifi(net)

    info("*** Stopping network\n")
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    stp = True if '-s' in sys.argv else False
    topology(stp)
```

Referencias

[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.