



**ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN**

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

TRABAJO FIN DE GRADO

TÍTULO:

SOFTWARE-DEFINED NETWORKING EN MININET WIFI

AUTORA: Manuela Calvo Barrios

TUTORA: Eva María Castro Barbero

CURSO ACADÉMICO: 2019/2020

Agradecimientos

Índice general

Agradecimientos

Resumen

Índice de figuras **III**

Índice de cuadros **V**

Lista de acrónimos y abreviaturas **VII**

1. Introducción **1**

1.1. Contexto y motivación 1

1.2. Objetivos 1

1.3. Estructura de la memoria 1

2. Estado del Arte **3**

2.1. Software-Defined Networking 3

2.1.1. Origen: la problemática de las redes actuales 3

2.1.2. SDN: concepto y arquitectura 5

2.2. OpenFlow 6

2.2.1. Puertos OpenFlow 6

Puertos físicos 7

Puertos lógicos	7
2.2.2. Switches de la red	7
2.2.3. Tablas de flujos	8
Cabecera	8
Acciones	9
Estadísticas	9
2.2.4. Tipos de mensajes OpenFlow	10
2.3. Mininet WiFi	10
3. Análisis script meshAP.py	11
3.1. Configuración del script meshAP.py	11
3.2. Topología inicial	13
3.3. Establecimiento de la conexión y configuración de las interfaces	14
3.4. Análisis de los flujos de datos con <code>pingall</code>	14
3.5. Capturando el tráfico OpenFlow entre el controlador y los <i>access points</i>	22
3.6. Topología completa resultante	30
4. Apéndice	33

Índice de figuras

3.1. Posición de los nodos de la red meshAP.	13
3.2. Topología de la red meshAP.	14
3.3. Consulta de ap1 al controlador	25
3.4. Respuesta del controlador a ap1	26
3.5. Consulta de ap2 al controlador	27
3.6. Respuesta de modificación del controlador a ap2	27
3.7. Respuesta del controlador con las acciones a instalar en ap2	28
3.8. Consulta de ap1 del ICMP echo request	28
3.9. Respuesta del controlador con la modificación del flujo en ap2	29
3.10. Respuesta del controlador con las acciones a instalar en ap1	29
3.11. Topología resultante de la red meshAP.	30

Índice de cuadros

4.1. Equivalencias entre las comorbilidades del IE y las comorbilidades del IC. . . .	33
---	----

Lista de acrónimos y abreviaturas

ARP Address Resolution Protocol

HTTP Hypertext Transfer Protocol

IDS Intrusion Detection System

IoT Internet of Things

IP Internet Protocol

NGN Next Generation Network

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

VoIP Voice over IP

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. 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 movilidad 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]

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]

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]

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]

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]

2.1.2. SDN: concepto y arquitectura

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]

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. [2]

En la red tradicional basada en sistemas distribuidos, los switches y routers controlan el enrutamiento 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]

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 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]

2.2. OpenFlow

Surgió como una alternativa a los protocolos empleados en los estudios de tráfico y redes que se hacían dentro de las universidades, por la facilidad que estas tienen para validar el adecuado funcionamiento de las nuevas tecnologías. 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]

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]

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. [4]

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. [4]

Igual que en las redes tradicionales, en OpenFlow existen diferentes tipos de puertos. En concreto en la versión 1.0 se usan los puertos físicos y los puertos virtuales, y en la versión 1.3 aparecen los puertos reservados. [4]

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" e interfaz virtual. [4]

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 se emplean puertos virtuales para el envío de paquete por inundación o para indicar cuál es el puerto de entrada al switch. [5]

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]

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. [4]

2.2.3. Tablas de flujos

OpenFlow basa sus operaciones en la llamada tabla de flujos. Esta tabla tiene principalmente 3 campos:

- 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 hay 12 campos, y en la versión 1.3 hay 40 campos.

La versión que emplea Mininet WiFi es OpenFlow 1.0. Sus principales mejoras respecto a las versiones anteriores son las siguientes: [4]

- 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 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.

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]

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]

Estadísticas

Las recopilan y almacenan los switches 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, porque no es el mismo tráfico el que hay a las 23:00h que a

las 03:00h, tampoco el que hay 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. [2]

2.2.4. Tipos de mensajes OpenFlow

2.3. Mininet WiFi

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 access points 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 es el siguiente:

```
1 #!/usr/bin/python
2
3 '''
4 This example shows on how to create wireless link between two APs
5 with mesh
6 The wireless mesh network is based on IEEE 802.11s
7 '''
8
9 from mininet.log import setLogLevel, info
10 from mn_wifi.link import wmediumd, mesh
11 from mn_wifi.cli import CLI
12 from mn_wifi.net import Mininet_wifi
13 from mn_wifi.wmediumdConnector import interference
14
15
16 def topology():
17     'Create a network.'
18     net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)
19
```

```

20 info('*** Creating nodes\n')
21 sta1 = net.addStation('sta1', mac='00:00:00:00:00:11',
22     position='1,1,0')
23 sta2 = net.addStation('sta2', mac='00:00:00:00:00:12',
24     position='31,11,0')
25 ap1 = net.addAccessPoint('ap1', wlans=2, ssid='ssid1',
26     position='10,10,0')
27 ap2 = net.addAccessPoint('ap2', wlans=2, ssid='ssid2',
28     position='30,10,0')
29 c0 = net.addController('c0')
30
31 info('*** Configuring wifi nodes\n')
32 net.configureWifiNodes()
33
34 info('*** Associating Stations\n')
35 net.addLink(sta1, ap1)
36 net.addLink(sta2, ap2)
37 net.addLink(ap1, intf='ap1-wlan2', cls=mesh, ssid='mesh-ssid',
38     channel=5)
39 net.addLink(ap2, intf='ap2-wlan2', cls=mesh, ssid='mesh-ssid',
40     channel=5)
41
42 info('*** Starting network\n')
43 net.build()
44 c0.start()
45 ap1.start([c0])
46 ap2.start([c0])
47
48 info('*** Running CLI\n')
49 CLI(net)
50
51 info('*** Stopping network\n')
52 net.stop()
53
54
55 if __name__ == '__main__':
56     setLogLevel('info')
57     topology()

```

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 puntos de acceso (*access points* (APs) en adelante) 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 *ap1* y *ap2* 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.

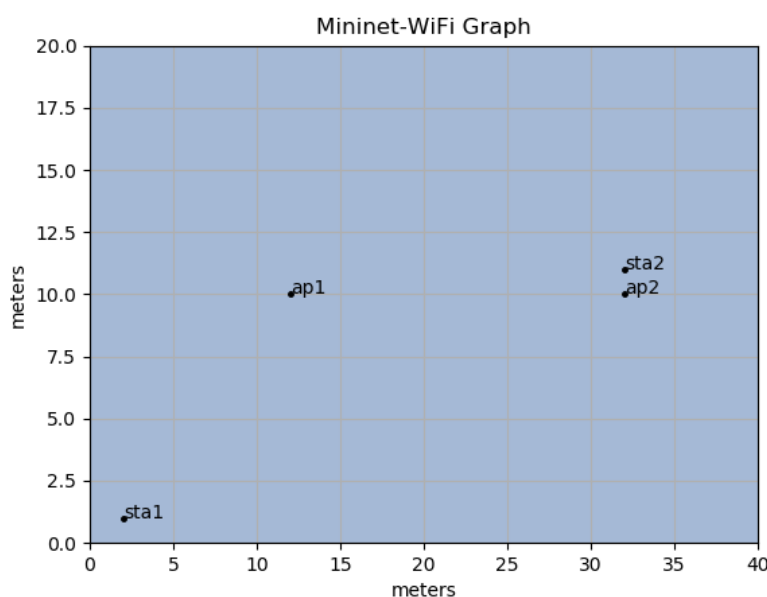


Figura 3.1: Posición de los nodos de la red meshAP.

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.

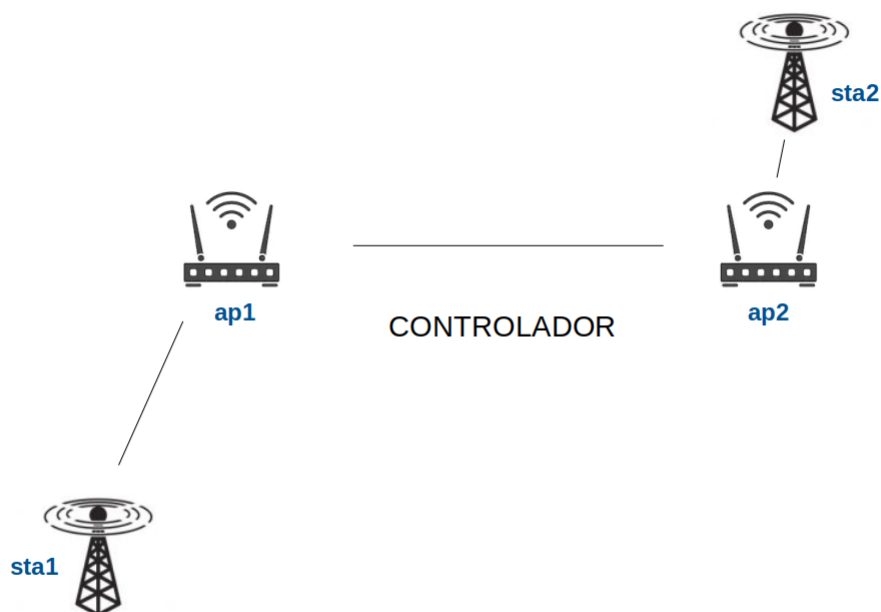


Figura 3.2: Topología de la red meshAP.

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 ori-

gen 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 dirección origen de ICMP 10.0.0.1 y destino ICMP 10.0.0.2, saldrán por la interfaz 'ap1-mp2' de ap1. No se corresponde a ninguna regla del controlador.

```
cookie=0x0, duration=5.051s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535, icmp,in_port='ap1-wlan1',vlan_tci=0x0000,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, con 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 las tablas de flujos.

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 ping, 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.

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.

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 por 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.

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


```

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

```

Figura 3.3: Consulta de ap1 al controlador

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

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

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

Figura 3.4: Respuesta del controlador a ap1

00:00:12,d1_dst=00:00:00:00:00:11, arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=output:'ap1-wlan1'

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,d1_src=00:00:00:00:00:12,d1_dst=00:00:00:00:00:11, arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=output:'ap2-mp2'

De igual manera, cuando sta1 envía el ICMP echo request hacia sta2, al llegar el paquete a la interfaz 'ap1-wlan1', ap1 consultará al controlador, como muestra la figura 3.8.

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.

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

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

Figura 3.5: Consulta de ap2 al controlador

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

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

Figura 3.6: Respuesta de modificación del controlador a ap2

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

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

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

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

577	93.3...	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN	
578	93.3...	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD	
579	93.3...	127.0.0.1	127.0.0.1	TCP	66	48942 → 6653 [ACK] Seq=6834 Ack=6636	

▶ Frame 577: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface 0
 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ Transmission Control Protocol, Src Port: 48942, Dst Port: 6653, Seq: 6718, Ack: 6556, Len: 116
 ▶ OpenFlow 1.0
 .000 0001 = Version: 1.0 (0x01)
 Type: OFPT_PACKET_IN (10)
 Length: 116
 Transaction ID: 0
 Buffer Id: 0xffffffff
 Total length: 98
 In port: 1
 Reason: No matching flow (table-miss flow entry) (0)
 Pad: 00
 ▶ Ethernet II, Src: 00:00:00_00:00:11 (00:00:00:00:00:11), Dst: 00:00:00_00:00:12 (00:00:00:00:00:12)
 ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
 ▶ Internet Control Message Protocol
 Type: 8 (Echo (ping) request)
 Code: 0
 Checksum: 0x2e85 [correct]
 [Checksum Status: Good]
 Identifier (BE): 22763 (0x58eb)
 Identifier (LE): 60248 (0xeb58)
 Sequence number (BE): 1 (0x0001)
 Sequence number (LE): 256 (0x0100)
 ▶ [No response seen]
 Timestamp from icmp data: Apr 26, 2020 19:31:41.000000000 CEST
 [Timestamp from icmp data (relative): 0.776392521 seconds]
 ▶ Data (48 bytes)

Figura 3.8: Consulta de ap1 del ICMP echo request

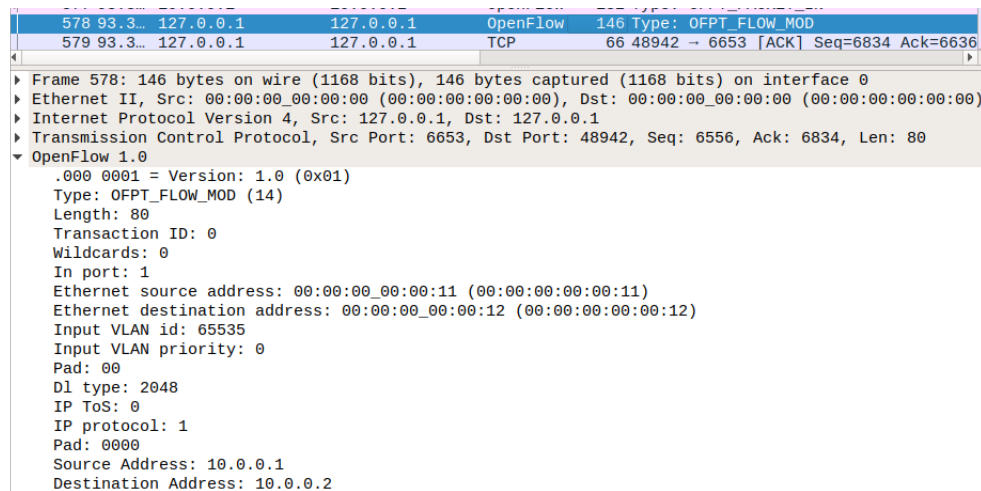


Figura 3.9: Respuesta del controlador con la modificación del flujo en ap2

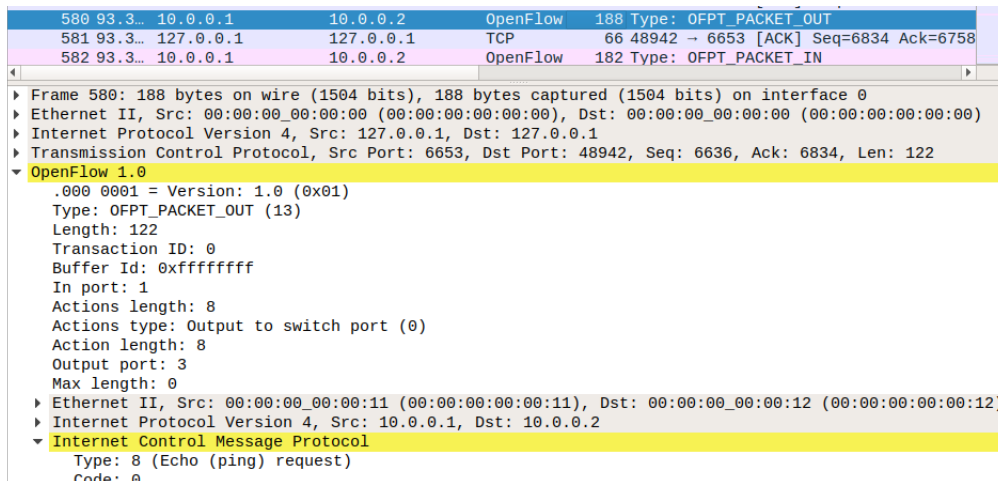


Figura 3.10: Respuesta del controlador con las acciones a instalar en ap1

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 el apartado 3.2. La topología resultante se muestra en el apartado 3.6.

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.

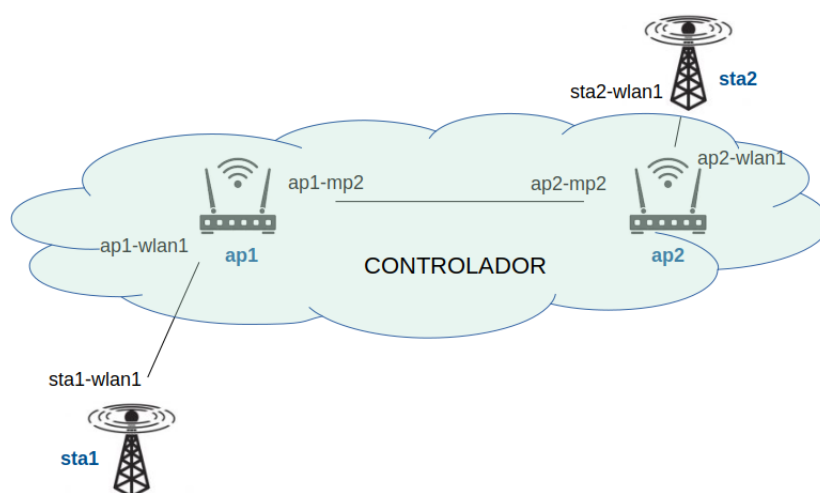


Figura 3.11: Topología resultante de la red meshAP.

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 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 autoequilibra. Esta es una de las ventajas de este tipo de redes definidas por software, su capacidad de autogestión y autoconocimiento.

Esta red analizada es sencilla y no tiene mucho sentido entonces que haya dos puntos de acceso diferentes. Sin embargo, cuando el tamaño de la red aumenta y hay varias estaciones

base que se comunican entre ellas, sí que estaría suficientemente motivado el uso de varios puntos de acceso para, como se ha indicado en el párrafo anterior, poder verificar que realmente la red es capaz de autogestionarse.

No obstante, dentro de la simplicidad que presenta meshAP 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

Apéndice

Cuadro ejemplo de hacer cuadro en latex. *Cuadro 4.1*

Comorbilidad del IE	Comorbilidad del IC	Nombre de la comorbilidad
1	2	Insuficiencia cardíaca congestiva
5	3	Enfermedad vascular periférica
7	12	Hemiplejía
9	6	Enfermedad pulmonar obstructiva crónica
16	17	SIDA
18	16	Tumor sólido con metástasis

Cuadro 4.1: Equivalencias entre las comorbilidades del IE y las comorbilidades del IC.

Bibliografía

- [1] Mireya Tovar Vidal. «Sistemas distribuidos». En: *Facultad de Ciencias de la Computación (BUAP)* (2015).
- [2] Ángel Leonardo Valdivieso Caraguay y col. «SDN: Evolution and Opportunities in the Development IoT Applications». En: *International Journal of Distributed Sensor Networks* (2014).
- [3] I. T. Haque y N. Abu-Ghazaleh. «Wireless Software Defined Networking: A Survey and Taxonomy». En: *IEEE Communications Surveys Tutorials* 18.4 (2016), págs. 2713-2737.
- [4] Open Networking Foundation. «OpenFlow Switch Specification Version 1.3.0». En: *ONF TS-006* (2012).
- [5] Open Networking Foundation. «OpenFlow Switch Specification Version 1.0.0». En: *ONF TS-001* (2009).