ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

SDN SOBRE REDES IEEE 802.11: SIMULACIÓN MEDIANTE MININET-WIFI

(SDN ON IEEE 802.11 NETWORKS: SIMULATION VIA MININET-WIFI)

Para acceder al Titulo de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Eduardo Gregorio Sáinz

Octubre - 2016

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Eduardo Gregorio Sáinz
Director del TFG: Roberto Sanz Gil
Título: "SDN sobre redes IEEE 802.11: simulación mediante MiniNet-Wifi"
Title: "SDN on IEEE 802.11 networks: simulation via MiniNet-Wifi"

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre):
Secretario (Apellidos, Nombre):
Vocal (Apellidos, Nombre):

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Agradecimientos

En primer lugar me gustaría agradecer a Roberto toda la ayuda que me ha prestado en la realización de este proyecto, sin importar el momento ni la hora, siempre ha estado dispuesto a ayudarme y a guiarme a la hora de realizarlo, y sobretodo para confiar en mí para permitirme hacer el Trabajo de Fin de Grado con él. Tambien me gustaría agradecérselo a mi familia y amigos por hacerme más llevadera la realización del mismo. Mencionar especialmente a mis compañeros de clase y amigos Javi, Nacho, Juan y Pablo con los que he pasado 4 muy buenos años y lo que nos queda.

Resumen

En éste trabajo se pretende analizar las redes SDN (Software-Defined Networking) así como el protocolo OpenFlow. Para llevar a cabo éste estudio, se utilizarán diferentes herramientas, como son: Mininet y Mininet-WiFi, ya que ésta última presenta novedades importantes como la incorporación de los modelos de movilidad. También se utilizará MiniEdit, se trata de la interfaz gráfica de Mininet que permite crear escenarios de forma más simple, por lo que se utilizará para la creación de alguno de esos escenarios, en cambio, otros escenarios se crearán mediante scripts de Python. Todo ésto irá acompañado del uso del controlador OpenDayLight, ya que se trata de un controlador con una interfaz gráfica muy avanzada y aporta diferentes ventajas con respecto a otros controladores. Por último, se empleará la herramienta Wireshark para la realización de capturas del tráfico OpenFlow.

Abstract

This project aims to analyse SDN (Software-Defined Networking) networks and the OpenFlow protocol. In order to carry out this study, different tools are going to be used, for example, Mininet and Mininet-WiFi. This last one has improved Mininet through some new and important changes like mobility models that make it essential in studies of this kind. So as to create some scenarios on an easier way MiniEdit, which is Mininet's graphical interface, is going to be used. Meanwhile, other scenarios are going to be created using Python scripts. In all this process, the OpenDayLight controller is going to be used, due to its advanced graphical interface and the fact that provides different advantages in comparison to other controllers. Finally, the Wireshark tool is going to be useful in order to analyse the OpenFlow traffic.

Motivación y objetivos

La motivación para la realización de éste trabajo está fundamentada en el crecimiento en importancia que está teniendo hoy en en día la arquitectura SDN, así como el protocolo OpenFlow. A día de hoy, las redes tienen una estructura muy estática, ésto provoca que tengan gran cantidad de limitaciones como son la inflexibilidad, el hardware dedicado y la creación de barreras a la optimización de la arquitectura de red y su capacidad de utilización, además, el tráfico con el incremento de la virtualización y la nube está tendiendo a ser entre servidores, y no como era antiguamente entre cliente y servidor, debido a ésto, y a otros factores como el aumento de ancho de banda y la flexibilidad que aporta SDN respecto a la arquitectura tradicional, ya que no está preparada para este tipo de tráfico debido a que son arquitecturas jerárquicas orientadas a tráfico Norte-Sur, se vió que la mejor estrategia para las redes era que la programación fuera de manera centralizada mediante software y no nodo a nodo.
OpenFlow se trata de un ejemplo, ya que no es el único protocolo dentro de la arquitectura SDN. Así pues, un software (los controladores) toman decisiones y son los encargados de reprogramar los caminos que seguirán los flujos de datos. Si alguno de los caminos tiene su capacidad de flujo completamente ocupada, el controlador tiene conocimiento de ello, de esta forma, puede seleccionar otro camino para el flujo de datos evitando así problemas de pérdidas de paquetes, encolamientos y demás problemas que pueden ocurrir si se usasen protocolos como OSPF y BGP.
Algunas de las ventajas que aportan tanto SDN como OpenFlow frente a las redes tradicionales son: potencial de revolucionar todas las redes, provocando una reducción de costes tanto en equipamiento como en operación, añadiendo mayor flexibilidad a la hora de controlar las redes y permite acelerar la introducción de nuevos servicios.
Los objetivos que persigue este trabajo es el estudio a diferentes niveles de SDN y OpenFlow, analizando el tráfico mediante Wireshark; como también aprender a manejar diferentes herramientas como Mininet y Mininet-WiFi junto al controlador OpenDayLight para la creación de escenarios con los que sea posible trabajar con el protocolo anteriormente mencionado.

Índice

Índice de figuras

Índice de tablas

1

1. Introducción

En este apartado se procederá a explicar la teoría necesaria para poder posteriormente entender el desarrollo de la parte más práctica del propio trabajo. Estará dividida en cuatro secciones en las que se explicará: SDN, OpenFlow, Mininet, Mininet-WiFi, MiniEdit, POX, OpenDayLight.

1.1. SDN

1.1.1. Introducción a SDN

Software-Defined Networking (SDN) es la separación en la gestión de los dispositivos de red del plano de control del plano de datos, en otras palabras, se trata de la separación del hardware y del software, y donde el control de la red es llevado a cabo por dispositivos denominados controladores [1] [2].

El objetivo principal que persigue SDN es facilitar la implementación e implantación de servicios de red de una manera determinista, dinámica, económica y escalable, evitando al administrador de red gestionar dichos servicios a bajo nivel [3].

En una red definida por software, un administrador de red puede darle forma al tráfico desde una consola de control centralizada sin tener que configurar los switches o routers individuales. Este administrador puede cambiar cualquier regla de los dispositivos cuando sea necesario, otorgando o quitando prioridad, o incluso hasta bloqueando tipos específicos de paquetes con un nivel de control muy detallado [2].

1.1.2. Arquitectura SDN

La arquitectura de SDN está formada por 3 capas que son accesibles desde interfaces de programación de aplicaciones (APIs-conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para poder ser utilizado por otro software como una capa de abstracción):

-La Capa de Aplicación: consiste en las aplicaciones destinadas a los usuarios finales que serán los consumidores de los servicios de comunicaciones SDN. Los usuarios finales utilizan servicios de comunicación SDN a través de la northbound API tales como REST, JSON, XML, entre otros,esta northbound API sirve para conectar el controlador SDN a los servicios y aplicaciones por encima; permite a los servicios y aplicaciones simplificar y automatizar las tareas de configuración, provisión y gestionar nuevos servicios en la red, ofreciendo a los operadores nuevas vías de ingresos, diferenciación e inovación, además de suplir las necesidades de las diferentes aplicaciones a través de la programabilidad de la red SDN. El límite entre esta capa y la siguiente, la Capa de Control, es atravesado por la northbound API. Se trata de las interfaces más críticas ya que como se menciona anteriormente soportan a gran cantidad de aplicacones y servicios por encima de ellas.

-La Capa de Control: proporciona una funcionalidad centralizada de control que supervisa el comportamiento de la red de datos a través de una interfaz abierta; permite a los desarrolladores de aplicaciones utilizar capacidades de red, pero abstrayéndose de su topología o funciones. En relación a esta capa debemos hacer referencia al controlador

SDN, ya que se trata de la entidad lógica de control encargada de traducir las peticiones del servicio SDN a las rutas de datos inferiores, dando a la capa de aplicación una visión abstracta de la red mediante estadísticas y posibles eventos. Se puede decir de los controladores que son el cerebro de este tipo de redes, ya que tienen control exclusivo sobre la forma de controlar y configurar los nodos de red para dirigir correctamente los flujos de tráfico; además de ésto, la arquitectura le permite generar un amplio rango de recursos del plano de datos, lo cuál ofrece el potencial de unificar y simplificar su configuración.

-La Capa de Infraestructura: está constituida por los nodos de red que realizan la conmutación y encaminamiento de paquetes. Proporciona un acceso abierto programable a través de la southbound API, como por ejemplo OpenFlow. Las southbound API facilitan el control en la red, permitiendo al controlador realizar cambios dinámicos de acuerdo a las demandas en tiempo real y las necesidades [1] [3] [4] [5] [6].

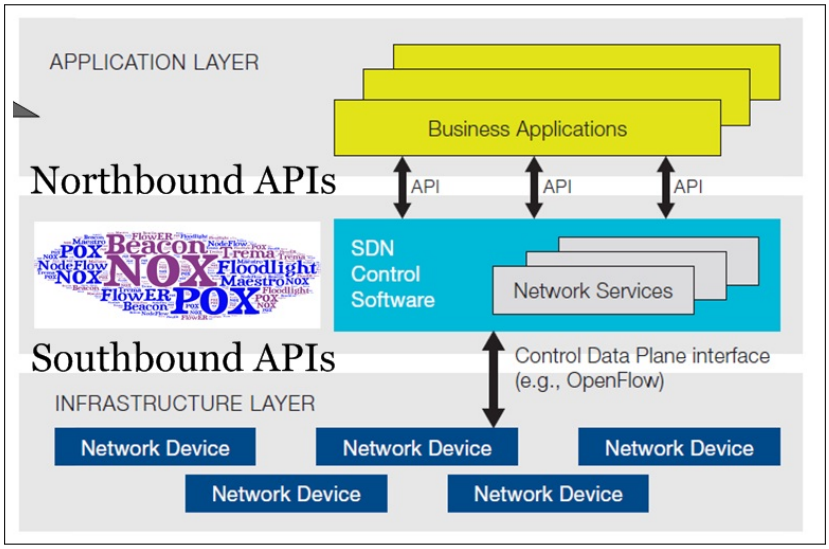


Figura 1: Arquitectura simplificada SDN

1.1.3. ¿Por qué SDN?

SDN surge con el objetivo de paliar las limitaciones que presentan las redes actuales, ya que la arquitectura SDN tiene el potencial para revolucionar todo el mundo de las redes, otorgando una manera flexible de controlarlas.

La principal motivación de las redes SDN es la reducción de los costes en equipamiento y operación de red, la primera de estas ventajas es conocida como Capex: permite la posibilidad de reutilizar el hardware existente, reduciendo así la necesidad de invertir en uno nuevo. Por otro lado, la reducción de operación en red se conoce como Opex: SDN permite el control algorítmico de la red y de los elementos de la misma, como switches y routers, que cada vez son más programables haciendo de una forma más sencilla la configuración y gestión de las redes. Además, de esta forma, se reduce la probabilidad de error humano ya que se reduce el tiempo de gestión por parte de los administradores.

Otros beneficios que aporta son la agilidad y flexibilidad ya que SDN permite a las organizaciones desplegar aplicaciones, servicios e infraestructuras rápidamente para alcanzar los objetivos propuestos por empresas.

Por útimo, añade nuevas oportunidades de negocio con sus clientes ofreciéndoles acceso a la información sobre el estado de la red y a la gestión de sus flujos de tráfico de forma más inteligente y eficiente en costes [3] [4].

1.2. Protocolo OpenFlow

Se trata de un protocolo que permite a un servidor decirle a los conmutadores de red dónde enviar paquetes. En una red convencional, cada dispositivo de red tiene software propietario que le dice qué hacer. Sin embargo, con OpenFlow se centralizan las decisiones de migración de paquetes, de modo que la red se puede programar independientemente de los conmutadores individuales y el equipo de centro de datos.

Un switch OpenFlow está formado al menos por tres partes, una o más Flow Tables, una Group Table, y un OpenFlow Channel para la conexión con el controlador.

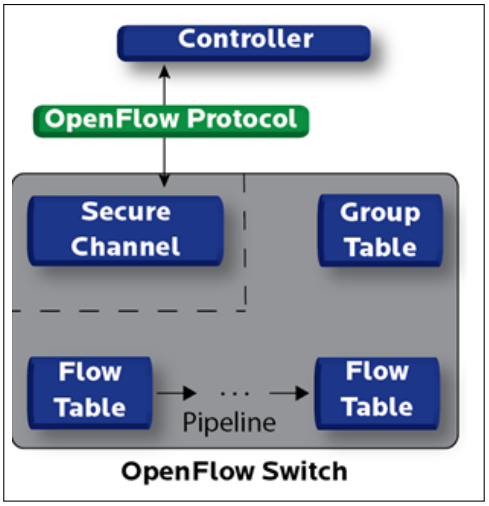


Figura 2: Arquitectura switch OpenFlow

- Tabla de flujos: con una acción asociada a cada entrada de la tabla, indicando al switch cómo debe procesar ese flujo. Ésta tabla de flujos consiste en una serie de entradas de flujo (flow entries).
Una entrada esta formada por los siguientes campos:

Match Fields	Counters	Intructions

Tabla 1: Flow Entry de una Flow Table.

- Match fields: cabecera y puerto.
- Counters: se actualiza cuando se encuentra una coincidencia.
- Instructions: para modificar el conjunto de acciones.

- Tabla de grupos: consiste en un grupo de entradas. Los grupos proveen una manera eficiente de indicar que el mismo conjunto de acciones deben ser llevadas a cabo por múltiples flujos. Por esta razón, es útil para implementar tanto multicast y unicast.

Group Identifier	Group Type	Counters	Action Buckets

Tabla 2: Flow Entry de una Group Table.

- Group Identifier: entero sin signo que identifica unívocamente al grupo.
- Group Type: determinan la semántica de grupo. Algunos de ellos son, por ejemplo: all (ejecutan todas las acciones de un grupo) y select (ejecuta una acción del grupo).

- Counters: se actualizan cuando los paquetes son procesados por un grupo.
- Action buckets: lista ordenada de acciones, donde cada acción contiene un conjunto de acciones a ejecutar y unos parámetros asociados.

- OpenFlow Channel: es la interfaz que conecta cada switch OpenFlow con un controlador. A través de ésta interfaz el controlador configura y maneja el switch, recibe eventos desde el switch, y envía paquetes fuera del switch. Entre el datapath y el OpenFlow Channel, la interfaz es implementación específica, sin embargo, todos los mensajes del OpenFlow Channel deben estar formateados de acuardo al protocolo OpenFlow. Normalmente está encriptado usando TLS, pero puede estar directamente sobre TCP [7] [8].

1.2.1. Matching

Cuando se recibe un paquete el OFS (OpenFlow Switch), el switch comienza haciendo una búsqueda en la primera tabla de flujo. Estas tablas se numeran empezando por cero, y basado en esta búsqueda realizará búsqueda en otras tablas de flujo. Si no encuentra coincidencia se actualiza el número de la tabla de flujo y salta a otra tabla, hasta encontrar la coincidencia; en caso de que no se encuentre coincidencia se devuelve al controlador [7] [8].

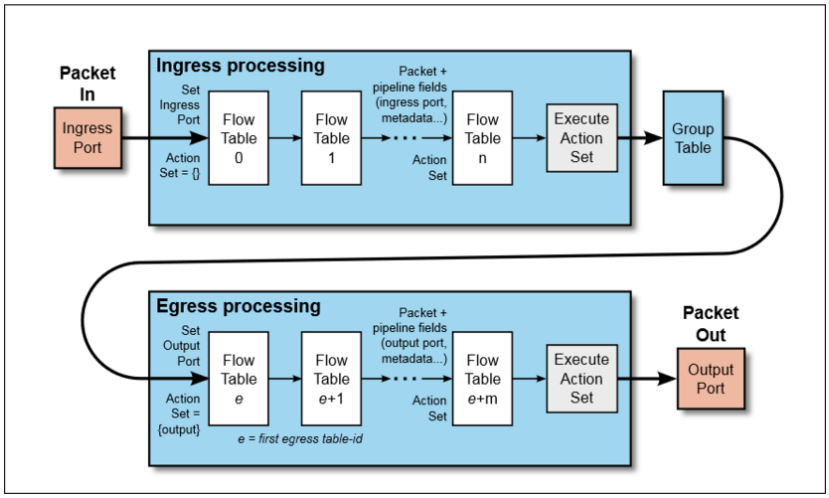


Figura 3: Proceso de matching

1.3. OpenFlow no es SDN

El protocolo OpenFlow constituye la base de las redes abiertas definidas por software (Software-Defined Networking, SDN) basadas en estándares. A menudo se apunta a OpenFlow como sinónimo de SDN, pero en realidad, es simplemente un elemento que forma parte de la arquitectura SDN. OpenFlow es un estándar abierto para un protocolo de comunicaciones que permite al plano de control interactuar con el plano de datos. Sin embargo, no es el único protocolo disponible o en desarrollo para SDN, entre otros se encuentran: MPLS-TP, BGP, OVSDB, XMPP; aunque sí está convirtiéndose en el modelo estándar de implementación de una red SDN [3].

1.4. Controladores OpenFlow

Un controlador OpenFlow ofrece una interfaz de programación para los conmutadores OpenFlow de tal forma que las aplicaciones de gestión pueden realizar tareas de gestión y ofrecer nuevas funcionalidades. Se trata del tipo de sistema operativo para la red, que aporta:

- Gestión del estado de la red.
- Un mecanismo de descubrimiento de dispositivos, topología y servicio.
- Un sistema de cálculo de ruta.
- Una sesión TCP entre el controlador y los agentes asociados en los elementos de la red.
- Un conjunto de APIs que exponen los servicios del controlador a las aplicaciones de gestión.

Toda comunicación entre las aplicaciones y cualquier dispositivo tiene que pasar a través de él. El protocolo OpenFlow conecta el software del controlador con los dispositivos de red de forma que éste pueda decir a los switches dónde enviar los paquetes. El controlador usa el protocolo OpenFlow para configurar los dispositivos de red y elegir el mejor camino para el tráfico. Por tanto, es el controlador central OpenFlow el que dicta el comportamiento de la red a partir de los requeriminetos de las aplicaciones. Entre otros controladores algunos de los más importantes son: NOX, POX, OpenDayLight(ODL), Floodlight, Beacon. En la tabla que se muestra a continuación (Cuadro 3) se realizará una breve comparativa entre las principales características de los controladores mencionados anteriormente [9] [10].

Características	Beacon	Floodlight	NOX	POX	ODL
Soporte OpenFlow	OFv1.0	OFv1.0	OFv1.0/v1.3	OFv1.0	OFv1.0/v1.3
Virtualización	Mininet OvS	Mininet OvS	Mininet OvS	Mininet OvS	Mininet OvS
Lenguaje de desarrollo	Java	Java	C++	Python	Java
Provee REST API	No	Si	No	No	Si
Interfaz gráfica	Web	Web	Python+ QT4	Python+ QT4/Web	Web
Soporte OpenStack	Linux MAC OS Windows Android Movil	Linux MAC OS Windows	Linux	Linux MAC OS Windows	Linux MAC OS Windows
Código Abierto	No	Si	No	No	Si
Tiempo en el mercado	Alto	Medio	Alto	Medio	Bajo
Rendimiento	Medio	Rápido	Lento	Rápido	Rápido
Documentación	Buena	Buena	Media	Pobre	Media

Tabla 3: Comparativa entre los controladores OpenFlow

Información tabla: OvS -> OpenvSwitch

Tras haber realizado la introducción a los controladores OpenFlow más importantes y, a su vez más utilizados, nos centraremos en los dos controladores sobre los que se ha investigado para la posterior elección, que son los siguientes:

- OpenDayLight
- POX

1.4.1. OpenDayLight

OpenDayLight(ODL) es un proyecto de Código Abierto cuyo objetivo es acelerar y aumentar la difusión de la innovación tanto en el diseño como en la implementación de SDN. Busca convertirse en una plataforma abierta que utilicen todas las empresas, evitando que las aplicaciones privadas disminuyan el crecimiento del mercado y, a su vez, reduzca los costes de desarrollo. Incorpora áreas como la plataforma del controlador o los protocolos de las aplicaciones de red, las interfaces de usuario, los switches virtuales o las interfaces físicas del dispositivo.
La principal ventaja de OpenDayLight es que elimina las barreras de adopción, ya que algunas organizaciones no quieren comprometerse con un fabricante. Al ser una plataforma abierta, las empresas disponen de la opción de optar por tecnologías de fabricantes diferentes que serán interoperables, estos son: Big Switch Networks, Broacade, Cisco, Citrix, Ericsson, IBM, Juniper Networks, Microsoft, NEC, RedHat y VMWare, ya que son los fundadores principales del proyecto.
En definitiva, ODL es un controlador de OpenFlow, el cuál, a su vez, es un protocolo que permite a un servidor decirle a los conmutadores de red a dónde enviar los paquetes. Además ODL, integra estándares abiertos y APIs abiertos para ofrecer una plataforma SDN que sea más programable, inteligente y adoptable.
Las principales distinciones respecto a SDN de OpenDayLight respecto a las opciones tradicionales son:

- Una arquitectura microservicios, en el que un microservicio es un protocolo o servicio en particular que un usuario quiere permitir dentro de su instalación del controlador ODL, por ejemplo: protocolo BGP, un servicio AAA(Autenticación, Autorización y Contabilidad).
- Proporciona soporte para una amplia gama de protocolos, no únicamente OpenFlow, también incluye SNMP, NETCONF, OVSDB, BGP, PCEP, LIS, entre otros.
- Soporte para el desarrollo de nuevas funcionalidades compuesto de protocolos y servicios de red adicionales.

OpenDayLight realiza las siguientes acciones:

- Control centralizado de los dispositivos físicos y virtuales en la red.
- Control de los dispositivos con estándares y protocolos abiertos.
- Proporciona abstracción de alto nivel de sus capacidades para que los ingenieros de redes y los desarrolladores puedan crear nuevas aplicaciones para personalizar la configuración y administración de redes.

Los casos de uso para SDN son los siguientes:

- Centralizado de monitorización de red, gestión y coordinación.
- Gestión proactiva de redes e ingeniería de tráfico.
- NUBE- gestionar tanto la superposición virtual y la capa base física debajo de ella.
- Encadenamiento de paquetes a través de las diferentes máquinas virtuales.

Interfaz gráfica OpenDayLight

Para abrir esta interfaz gráfica lo primero que se debe hacer es entrar en un navegador y con la URL correcta se podrá acceder a esta interfaz, estos pasos para acceder y entrar en la interfaz se explicarán posteriormente en el apartado de Instalación. Ahora se va a explicar las diferentes partes que forman la interfaz gráfica, que son las siguientes:

- Topology
- Nodes
- Yang UI
- Yang Visualizer

· **Topology:** en esta sección se puede observar la topología que es emulada por Mininet. Para que se pueda ver la topología es necesario enviar un ping a todos los hosts ya que sino el controlador no es capaz de mostrar toda la red. Cada vez que se pulse sobre el botón Reload la disposición tanto de los nodos como de los switches o routers sera diferente.

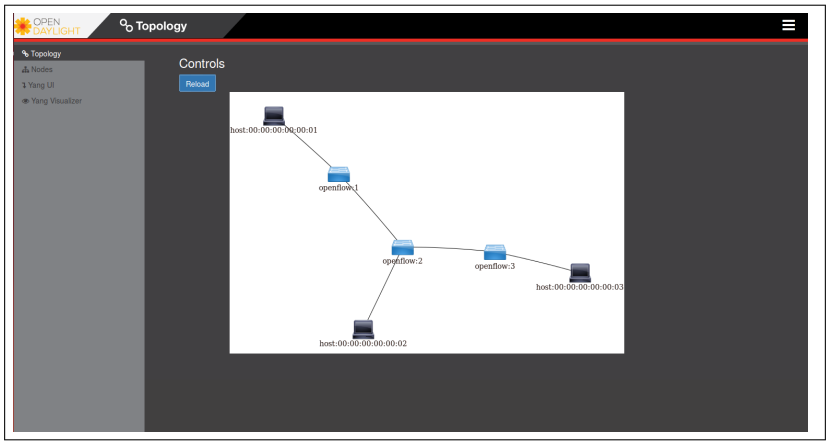


Figura 4: Sección Topology de la interfaz gráfica de OpenDayLight

· **Nodes:** en esta otra sección se puede ver la información de cada switch que forma la red. Dentro de la tabla que aparece se puede encontrar un apartado denominado "node connectors", donde se encuentra la información de cada parte del switch que se haya seleccionado.

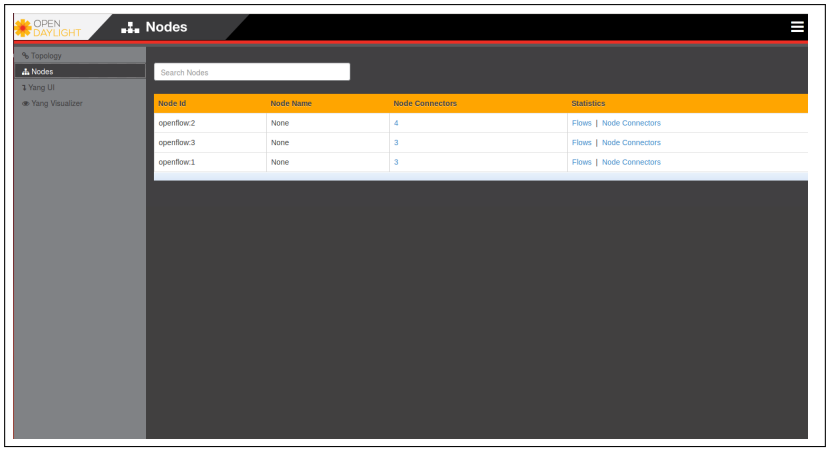


Figura 5: Sección Nodes de la interfaz gráfica de OpenDaayLight

· **Yang UI:** Yang se trata de una estructura de modelado de datos. Provee funcionalidades en los switches de SDN. La Yang UI de ODL es un cliente REST(Representational State Transfer) para la contrucción y envío de las peticiones REST al almacén de datos de ODL. Se puede utilizar la Yang UI para obtener información del almacén de datos o para construir comandos y modificar la información de este almacén, cambiando así la configuración de la red. Dentro de la Yang UI se dispone de diferentes opciones:API, HISTORY, COLLECTION, PARAMETERS. Si se pulsa en API se podrá ver todas las APIs que estén disponibles, pero no todas ellas funcionarán ya que esto dependerá de si se tienen instaladas esas funcionalidades o no. Una que si funcionará será Ïnventory API", pinchando en ella y desplegando las opciones que muestra se observará un inventario de información de la red, como son las estadísticas, los puertos, los nodos, entre otros. Entendiendo el modelo de datos de Yang y aprendiendo a usarlo es una de las claves para entender SDN a través del uso de OpenDayLight.

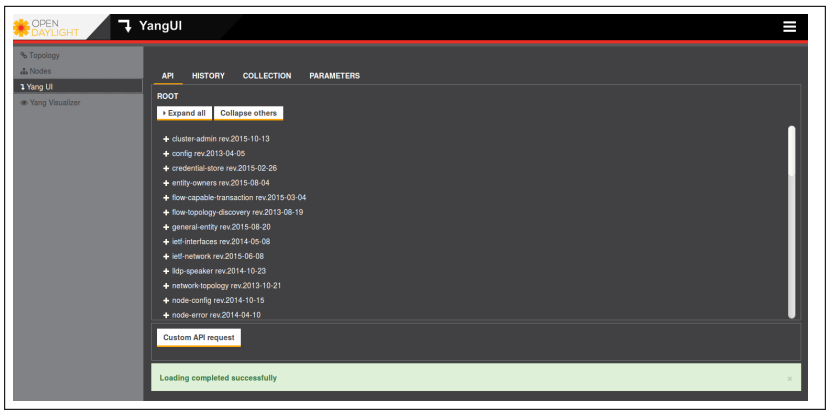


Figura 6: Sección YangUI de la interfaz gráfica de OpenDaayLight

· **Yang Visualizer:** esta sección solo está disponible en las dos últimas versiones de OpenDayLight, pero como la versión que se va a utilizar en el proyecto es Berillium, la última versión disponible, sí se dispondrá de ello. Esta sección lo que aporta es una característica más visual del Yang UI.

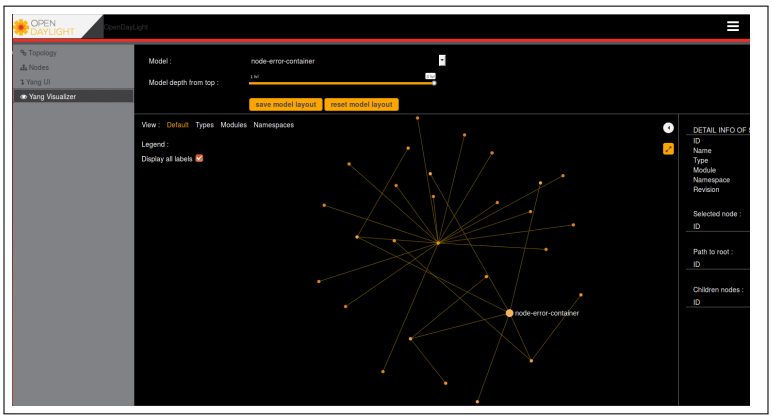


Figura 7: Sección Yangvisualizer de la interfaz gráfica de OpenDaayLight

Todas estas secciones servirán para conocer de una mejor forma el tráfico OpenFlow entre los nodos, así como la red en general [10] [11] [12].

1.4.2. POX

POX es un controlador SDN fácil de usar, desarrollado en Python que permite programar y controlar switches que soportan el protocolo OpenFlow. A su vez, los componentes POX son programas Python que implementan las funciones de red y pueden ser invocados cuando se inicia POX. Se trata del hermano de NOX. En su esencia, es una plataforma para el desarrollo y creación de prototipos de software de control de la red. Además de ser un marco para interactuar con los interruptores de OpenFlow, se usa para construir la disciplina emergente de SDN. Se encuentra bajo un desarrollo activo. Algunas de las características de POX son:

- Interfaz realizada en Python.
- Componentes de la muestra reutilizables para la selección de la ruta, el descubrimiento de topología, etc.
- Ejecutable en cualquier lugar.
- Dirigido tanto a Linux, Mac OS y Windows.
- Compatible con las mismas herramientas de visualización que la GUI de NOX.
- Se comporta mejor en comparación con las aplicaciones de NOX escritas en Python [13].

1.5. Mininet y Mininet-Wifi

1.5.1. Mininet

Mininet es un emulador de red o de forma más precisa, un sistema de emulación de red basado en la manipulación y organización (network emulation orchestration system). Permite virtualizar hosts, switches, routers y enlaces en un núcleo (kernel) de Linux. Usa una virtualización muy ligera para realizar un sistema que simula una red completa, ejecutando el mismo kernel, sistema y código de usuario. Un host de Mininet se comporta como una máquina real. Los programas que se pueden ejecutar envián paquetes a través de la red, que parece una interfaz de red con su correspondiente velocidad y retardo. Los paquetes son procesados por un switch o router que parece un switch o router Ethernet real. Resumiendo, los hosts, switches, enlaces y controladores virtuales de Mininet son reales, únicamente que se ha empleado software en vez de hardware, y el comportamiento es muy similar.

Mininet nos aporta una serie de beneficios:

- Velocidad: ya que crear una red es relativamente fácil y no requiere de más de unos segundos.
- Posibilidad de crear infinitos tipos de topologías.
- Puede ejecutar programas de los que se disponga en la máquina Linux, como puede ser, por ejemplo, Wireshark.
- Permite personalizar los paquetes enviados (customize packet forwarding): los switches de Mininet pueden ser programados usando el Protocolo OpenFlow.
- Diferentes formas donde ejecutar Mininet: en un portátil, en un servidor, en una máquina virtual o en un sitema nativo de Linux.
- Compartir resultados.
- Fácilmente de usar: mediante scripts de Python se pueden crear y ejecutar diferentes experimentos de Mininet.

- Mininet es un programa Open-Source.
- Está bajo un activo desarrollo y mejora contínua.

Aunque Mininet aporta gran cantidad de beneficios también tiene una serie de limitaciones, entre otras, las más importantes son las siguientes:

- Usa un único núcleo de Linux para todos los hosts virtuales. Por lo que no se puede ejecutar software que dependa de Windows u otros sistemas operativos.
- Necesidad de utilizar un controlador OpenFlow independiente.
- No posee una noción de tiempo virtual, ésto quiere decir que las medidas estarán basadas en tiempo real y por lo tanto los resultados serán más rápidos que en la situación real [14].

1.5.2. Mininet-WiFi

Se trata de una herramienta del emulador de SDN de Mininet. Los desarrolladores de Mininet-WiFi aumentaron las funcionalidades de Mininet añadiendo estaciones y puntos de acceso inalámbricos basados en un controlador 80211_hwsim. Además, añaden clases para soportar estos dispositivos inalámbricos en los escenarios de Mininet, también se añaden atributos como la posición y el movimiento relativo de las estaciones móviles y los puntos de acceso. Mininet-WiFi extiende el código base de Mininet, añadiendo o modificando clases y scripts. Por lo tanto, Mininet-WiFi añade nuevas funcionalidades y conserva aquellas que tenía Mininet original [15].

1.6. MiniEdit

MiniEdit se trata de un editor GUI para Mininet. Esta herramienta es capaz de crear y ejecutar diferentes simulaciones de red, como a su vez, configurar los elementos de esta red y guardar la topología.

La interfaz de MiniEdit presenta en su parte izquierda los diferentes elementos que se pueden utilizar, y también un menú en su parte superior.

Figura 8: Interfaz de MiniEdit

Los elementos que presenta la herramienta por orden de apariencia de arriba a abajo son los siguientes:

- Select
- Host
- Switch para OpenFlow
- Legacy Switch para Ethernet
- Legacy Router
- NetLink
- Controller
- Run o Stop

Select: permite mover cualquier nodo dentro de la interfaz. También permite entrar en las características de cada uno de estos nodos, como a su vez, eliminarlos.

Host: realizan la función de host. Es posible configurar cada host pulsando con el click derecho y seleccionando Properties del menú.

Switch para OpenFlow: la herramienta crea interruptores OpenFlow. El objetivo de estos switches es que sean conectados a un controlador. Opera de la misma forma que el anterior elemento descrito, pudiendo cambiar las propiedades mediante el botón derecho del ratón.

Legacy Switch para Ethernet: crea un conmutador Ethernet de aprendizaje con la configuración predeterminada. El switch opera de forma independiente, sin un controlador. El conmutador no puede ser configurado con el Spanning Tree, por lo que no se conectan los switches existentes en bucle.

Legacy Router: crea un router básico que opera de manera independiente sin un controlador. Es, básicamente, solo un host con el reenvío IP habilitado. Este router no se puede editar desde la interfaz gráfica de usuario de MiniEdit.

NetLink: crea enlaces entre nodos. Para ello, basta con seleccionar un nodo y arrastrar hasta el nodo con el que se quiere conectar. Es posible configurar las propiedades de cada enlace. Como también, se puede simular la caída de un enlace mediante las opciones de Link Up y Link Down durante la ejecución.

Controller: crea un controlador. Es posible la creación de más de un controlador. Por defecto, MiniEdit crea un controlador OpenFlow que implementa el comportamiento de un interruptor de aprendizaje. El usuario puede configurar las propiedades de cada controlador.

Run o Stop: inicia o detiene el escenario de simulación [16].

2

2. Instalación

En esta sección se van a indicar los diferentes procesos que hay que llevar a cabo para la instalación tanto de Mininet, Mininet-WiFi y los contralodres necesarios. También los diferentes elementos que se necesitarán para tomar medidas.
La instalación se llevó a cabo en una máquina Linux con una distribución Ubuntu 14.04.1 y un kernel 4.2.0-27-generic.

2.1. Instalación Mininet

A continuación se explican los diferentes pasos que hay que seguir a la hora de instalar la herramienta Mininet:

```
sudo apt-get install mininet
```

Con este comando se instalarán los núcleos para Mininet.

```
sudo mn -c
```

Este paso permite matar cualquier controlador que Mininet pueda haber activado anteriormente.

```
sudo apt-get install git
```

Con ésto se instala git para que Mininet pueda ser descargado desde Github y crear una estructura de ficheros en nuestra máquina.

```
sudo git clone git://github.com/mininet/mininet
```

Se descarga el git. Con los siguientes comandos lo que se hará es cambiar a la última versión de Mininet.

```
cd mininet
git tag # list available versions
git checkout -b cs244-spring-2012-final

cd ..
```

El siguiente paso es el más importante ya que permite instalar todos los elementos que ofrece Mininet. Si este paso no es llevado a cabo Mininet no podría conectarse correctamente con el controlador y Wireshark no funcionará.

```
mininet/util/install.sh -a
```

En caso de que este comando no funcione utilizar el siguiente:

```
mininet/util/install.sh -a Ubuntu trusty i386 Ubuntu
```

Con esto ya se tendrá instalado Mininet.

2.2. Instalación Mininet-WiFi

Para la instalación de Mininet-WiFi únicamente es necesario seguir una serie de sencillos pasos:

```
sudo apt-get update
sudo apt-get install git make
git clone https://github.com/intrig-unicamp/mininet-wifi
cd mininet-wifi
sudo util/install.sh -Wnf3vpw
```

En este último paso lo que se está realizando es la elección de los paquetes que queremos instalar, son los siguientes:

- W: install Mininet-WiFi dependencies
- n: install Mininet dependencies + core files
- f: install OpenFlow
- 3: install OpenFlow 1.3
- v: install Open Vswitch
- p: install POX OpenFlow Controller
- w: install Wireshark

Con esto ya se tendrá instalado Mininet-WiFi, no es necesario instalar Mininet, ya que todas las caracterrísticas de éste están incluidas en Mininet-WiFi [15].

2.3. Instalación de MiniEdit

MiniEdit viene incluido en Mininet, lo único que se debe saber es cómo llegar a ello. Para ello, basta con seguir los pasos indicados a continuación:

```
sudo su
cd mininet-wifi
cd mininet
cd examples
./miniedit.py
```

La forma que arriba se indica es entrando a través de Mininet-WiFi, en caso que se quiera entrar a través de Mininet, la única diferencia sería quitar la linea de código:

```
cd mininet-wifi
```

2.4. Instalación OpenDayLight

Para la instalación del controlador OpenDayLight primero se debe instalar Java ya que este controlador es un programa Java. Para instalar Java se deben usar los siguientes comandos:

```
sudo apt-get update
sudo apt-get install default-jre-headless
```

Tras esto hay que editar el fichero ".bashrc"

```
nano ~/.bashrc
export JAVA_HOME=/usr/lib/jvm/default-java
```

Una vez se hayan realizado estos pasos se debe ejecutar el fichero:

```
source ~/.bashrc
```

En el momento que ya se haya instalado Java se podrá proceder a la instalación de OpenDayLight:

Se empezará por descargar la ultima versión de ODL, en este caso, Beryllium:

```
wget
https://nexus.opendaylight.org/content/groups/public/org/opendaylight/integration
/distribution-karaf/0.4.0-Beryllium/distribution-karaf-0.4.0-Beryllium.tar.gz
```

Una vez descargado se extrae el archivo:

```
tar -xvf distribution-karaf-0.4.0-Beryllium.tar.gz
```

Esto creará una carpeta llamada "distribution-karaf-0.4.0-Beryllium"que es la que contiene el software y los plugins de OpenDayLight. Karaf es un container tecnológico que permite a los desarrolladores situar todo el software requerido en una misma carpeta, permitiendo instalar o reinstalar ODL de una forma más fácil, ya que como se ha mencionado anteriormente, todo se encuentra en una misma carpeta. Ahora se procede a arrancar ODL:

```
cd distribution-karaf-0.4.0-Beryllium
./bin/karaf
```

Tras esto el controlador OpenDayLight estará ejecutándose.

Figura 9: OpenDayLight en ejecución

Ahora se deben instalar las características requeridas para la prueba de OpenDayLight y la GUI de OpenDayLight, los siguientes comandos hay que introducirlos en la ventana de comandos del usuario de ODL:

```
feature:install odl-restconf odl-l2switch-switch odl-mdsal-apidocs
odl-dlux-all
```

Las características que se han instalado son las siguientes:

- odl-restconf: permite el acceso a RESTCONF API
- odl l2switch-switch: provee funcionalidades de red similares a un switch Ethernet.
- odl-mdsal-apidocs: permite el acceso a la Yang API.
- odl-dlux-all: la interfaz de usuario gráfica de OpenDayLight.

Para ver las posibles características disponibles a instalar basta con el comando:

```
feature:list
```

Para ver las características ya instaladas:

```
feature:list --installed
```

Tras tenerlo instalado, se ha de buscar la dirección en la que se encontrará nuestro controlador, para ello:

```
ip addr show
```

Tras obtener la dirección (ésta dirección es la IP local), se accederá al navegador, para ello, en este caso:

```
http://192.168.1.36:8181/index.html#/login
```

Ya solo falta escribir en el usuario y en la contraseña: admin, ya que viene por defecto [17].

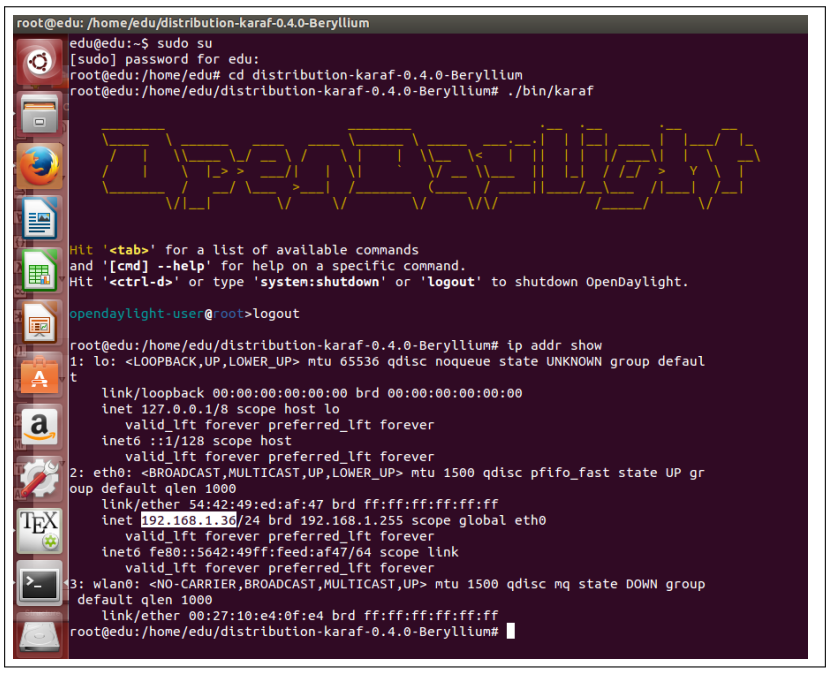


Figura 10: OpenDayLight: dirección en el navegador

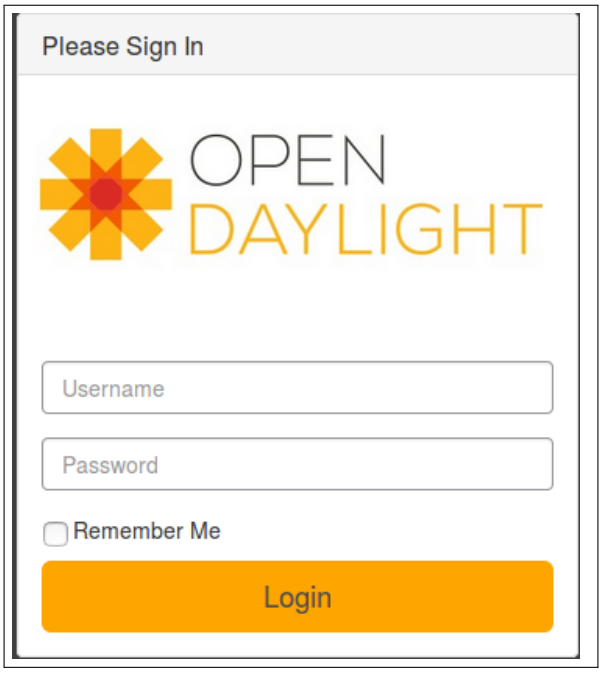


Figura 11: OpenDayLight: pantalla de admisión al controlador

2.5. Instalación Wireshark

Para la instalación del Wireshark basta con escribir en la ventana de comandos el siguiente comando:

```
sudo apt-get install wireshark
```

3

3. Introducción al uso de Mininet-Wifi

En este capítulo se va a tratar de realizar una introducción a las diferentes funcionalidades que presenta la herramienta, para ello se va a utilizar un pequeño tutorial formado por 4 partes [15]:

- Un punto de acceso.
- Múltiples puntos de acceso.
- Python API y scripts.
- Movilidad.

3.1. Un punto de acceso

En esta primera parte se muestra cómo crear escenarios simples así como la forma de capturar tráfico en una red wireless Mininet-WiFi. La topología esta formada por un punto de acceso wireless, se trata de un switch conectado a un controlado y dos estaciones también inalámbricas que en este caso son hosts.

3.1.1. Captura de tráfico de control inalámbrico

Lo primero que se hará será arrancar Wireshark.

```
cd mininet-wifi
wireshark &
```

Tras esto, se iniciará Mininet-WiFi con un escenario por defecto.

```
cd mininet-wifi
sudo mn --wifi
```

Ahora se activará la interfaz en la que se va a capturar.

```
sh ifconfig hwsim0 up
```

Se envía un ping desde la estación 1 (sta1) a la estación 2 (sta2).

```
sta1 ping sta2
```

Si lo que se quiere ver son los paquetes de ICMP encapsulados se capturará en la interfaz creada anteriormente; por el contrario si lo que se quiere es capturar los paquetes de OpenFlow se debe capturar en la interfaz de Loopback. Para comprobar si los flujos han sido creados basta con el comando:

```
dpctl dump-flows
```

Para parar la primera parte del tutorial:

```
exit
sudo mn -c
```

3.2. Múltiples puntos de acceso

En esta parte se va a crear un escenario de red con 2 o más puntos de acceso inalámbricos. Con el siguiente comando se creará una topologia de red en la que se dispondrá de tres puntos de acceso inalámbricos y una estación conectada a cada punto de acceso:

```
sudo mn --wifi --topo linear,3
```

Como resultado se ve cómo la red es establecida y cómo cada estación es relacionada con cada punto de acceso.

Se puede verificar la configuración usando los comandos *net* y *dump*. Si se quiere saber cuáles son los puntos de acceso visibles para cada estación se usará el comando siguiente:

```
sta1 iw dev sta1-wlan0 scan | grep ssid
```

Si lo que se quiere saber es con qué punto de acceso esta conectada nuestra estación se utilizará el comando:

```
sta1 iw dev sta1-wlan0 link
```

3.2.1. Escenario simple de movilidad

Con el comando `iw` se puede cambiar la conexión entre los puntos de acceso y las estaciones. Ésto no es recomendable cuando no son escenarios estáticos. Y en caso de que la obtención de esta información sea durante la ejecución será mejor utilizar el comando `info` del que se hablará más adelante.

```
sta1 iw dev sta1-wlan0 disconnect
sta1 iw dev sta1-wlan0 connect ssid_ap2
```

Lo que se ha hecho es cambiar la conexion de la estación, por lo que ahora estará conectada al punto de acceso 2. Esto se puede verificar con el comando *link* que se ha usado anteriormente.

3.2.2. Flujo OpenFlow en escenarios de movilidad

Ahora se va a ver cómo maneja el controlador este escenario. Para ello, se va a generar tráfico entre la estación 1 y la estación 3. Para saber las direcciones IP basta con escribir el comando *dump*. Una vez se sabe las direcciones IP de las estaciones se va a abrir una ventana externa para una de las estaciones:

```
xterm sta3
```

Desde esta nueva ventana se enviará un ping a la estación 1:

```
ping 10.0.0.1
```

Ahora se abrirá Wireshark para observar el flujo de OpenFlow, como recordatorio decir que hay que capturar en la interfaz de loopback. Para filtrar los paquetes usar 'of'.

Tras esto, en el cliente Mininet, se puede comprobar los flujos en cada punto de acceso:

```
dpctl dump-flows
```

Se puede observar flujos en los puntos de acceso 2 y 3, pero no en el 1, esto se debe a que ahora la estación 1 está conectado al punto de acceso 2 por lo que todo el tráfico va por los puntos de acceso 2 y 3. Si lo que se quiere es que haya flujo en los tres basta con volver a la configuración que se tenía inicialmente ya que ésta es la que se había modificado. Se debe tener en cuenta que al hacer ésto el controlador no es capaz de detectar las situaciones en las que una estación se mueve y cambia de punto de acceso por lo que se debe borrar los flujos con el siguiente comando:

```
dpctl del-flows
```

Para evitar esta situación se debe usar un controlador más avanzado como puede ser OpenDayLight.

Por último se va a detener este escenario para pasar a la siguiente parte del tutorial:

```
exit
sudo mn -c
```

3.3. Pyhton API y scripts

Mediante una API de Python los usuarios pueden crear scripts a través de los que se crean topologías. En la carpeta de examples dentro de Mininet-WiFi se podrán encontrar diferentes ejemplos.

3.3.1. Métodos para puntos de acceso y estaciones básicas

En un escenario se puede crear una estación en un script de Python de la siguiente forma:

```
net.addStation('sta1')
```

Con esto se añade una estación con todos los parámetros por defecto.

Para crear un punto de acceso con todos los valores por defecto y con SSID:

```
net.addBaseStation('ap1', ssid='new_ssid')
```

Una vez creado tanto el punto de acceso como la estación, se necesita crear el enlace entre ambos, para ello:

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

Si lo que se quiere es crear escenarios más complejos, se pueden añadir parámetros como la MAC, la IP, localización en 3D, contraseña, tipo de encriptación, entre otros. Ésto se puede realizar tanto para un punto de acceso como para una estación. Por ejemplo:

```
net.addStation( 'sta1', passwd='123456789a', encrypt='wpa2',
mac='00:00:00:00:00:02', ip='10.0.0.2/8', position='50,30,0' )
```

En los enlaces también se puede añadir características como la tasa de error o el ancho de banda máximo y el retraso:

```
net.addLink( ap1, sta1, bw='11Mbps', loss='0.1%', delay='15ms')
```

Para realizar las asociaciones de control en una red estática se puede usar el método associationControl. Por ejemplo:

```
net.associationControl('ssf')
```

3.3.2. API clásica de Mininet

Mininet-WiFi soporta los estándares sobre todo tipo de nodos (switches, hosts y controladores) de Mininet.

Para añadir un host:

```
net.addHost('h1')
```

Se tiene que tener en cuenta que cuando se habla de añadir un host hace referencia a añadir un host con una interfaz Ethernet, mientras que anteriormente cuando se refería a estaciones hacía referencia a un host con una interfaz inalámbrica.

Para añadir un switch:

```
net.addSwitch('s1')
```

En este caso, cuando se refiere a un punto de acceso es haciendo referencia a un switch con una interfaz inalámbrica y un número cualquiera de interfaces Ethernet (esto dependerá de la versión de OpenvSwitch).

Para añadir un controlador:

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

Con todo ésto ahora mismo ya se podría crear una topología compleja con hosts, switches, estaciones, puntos de acceso y múltiples controladores.

3.3.3. Red Mininet-WiFi y la posición de los nodos

Es posible añadir información relacionada con la posición de los nodos. Por ejemplo:

```
ap1 = net.addBaseStation( 'ap1', ssid= 'ssid-ap1', mode= 'g', channel=
                '1', position='10,30,0', range='20' )
```

3.3.4. Trabajando con Mininet-WiFi durante la ejecución

Los scripts pueden ser ejecutados tanto corriendo directamente el script:

```
sudo ./xxxxxxx.py
```

Como siendo parte de un comando de Python:

```
sudo python xxxxxxxx.py
```

Es muy posible que una vez que se tenga creado nuestro script no deje ejecutarlo debido a que no tiene permisos, por lo que previamente hay que otorgarle los permisos necesarios, mediante las instruccioes siguientes:

```
chmod 777 xxxxxxxx.py
```

Cuando el escenario está corriendo se puede recoger información de la red desde la ventana de comandos, pero también del interprete de Python , y a su vez, se pueden realizar cambios en la configuración de los nodos.

Algunos de los datos que se pueden obtener son:

- La posición:

```
position ap1
```

- La distancia:

```
distance ap1 sta2
```

- Información(número de asociaciones, potencia,SSID)

```
info ap1
```

3.3.5. Intérprete de Python para la recolección de datos

Para obtener el rango de un punto de acceso o una estación:

```
py ap1.py
```

Para ver qué estación está conectada con un punto de acceso:

```
py ap1.associatedStations
```

Para ver el número de estaciones asociadas a un punto de acceso:

```
py sta1.nAssociatedStations
```

Para ver que punto de acceso está asociado a una estación:

```
py sta1.associatedAp
```

Otros datos que se pueden obtener de esta forma son la potencia, el SSID, el canal y la frecuencia de cada nodo inalámbrico.

3.3.6. Realización de cambios durante la ejecución

Para cambiar el punto de acceso al que está conectado:

```
py sta1.moveAssociationTo('sta1-wlan0','ap1')
```

Cambiar la coordenada del punto de acceso o de la estación:

```
py sta1.moveStationTo('40,20,20')
```

Para cambiar el rango:

```
py sta1.setRange(100)
```

3.4. Movilidad

Posiblemente, la característica más importante que ofrece Mininet-WiFi con respecto a Mininet sea el soporte sobre la movilidad de las estaciones. En esta parte, se va a ver cómo manejar y crear un escenario donde las estaciones se muevan a lo largo del espacio y vayan cambiando de puntos de acceso a los que se conectan, basándose en la cercanía.

3.4.1. Movilidad y la API de Python

Para realizar movimiento en línea recta hay que usar los métodos `net.StartMobility` y `net.mobility`. Por ejemplo para mover de un punto a otro una estación a lo largo de 60 segundos:

```
net.startMobility( startTime=0 )
net.mobility( 'sta1', 'start', time=1, position='10,20,0' )
net.mobility( 'sta1', 'stop', time=59, position='30,50,0' )
net.stopMobility( stopTime=60 )
```

Si lo que se quiere es que el movimiento sea aleatorio se pueden utilizar alguno de los siguientes métodos: `RandomWalk`, `TruncatedLevyWalk`, `RandomDirection`, `RandomWayPoint`, `GaussMarkov`, `ReferencePoint`,`TimeVariantCommunity`. Por ejemplo:

```
net.startMobility(startTime=0, model='RandomDirection', max_x=60,
            max_y=60, min_v=0.1, max_v=0.2, AC='llf')
```

En este ejmplo se puede observar un parámetro AC que es el encargado de las asociaciones de control: - iif (Least-Loaded-First) - ssf (Strongest-Signal-First)

3.4.2. Test con iperf

Para ver cómo responde el sistema al tráfico realizaremos un ping, cuando ya se haya realizdo ese ping. Se inicia un servidor iperf en la estación:

```
sta1 iperf --server &
```

Tras ésto, se abrirá una ventana de comado en un host o en una estación con el comando `xterm` ya mencionado.

Ahora se iniciará un cliente iperf en esta ventana nueva que se ha abierto con `xterm`:

```
iperf --client 10.0.0.2 --time 60 --interval 2
```

Se puede observar la salida de iperf y como la estación se mueve a través del gráfico. Cuando esta pasa de un punto de acceso a otro el tráfico se para.
Para que vuelva a haber tráfico es necesario limpiar las tablas de flujo, ya que sino el controlador puede funcionar de manera incorrecta.
Para ello, basta con el siguiente comando:

```
dpctl del-flows
```

3.5. Propagación

En los escenarios de Mininet-WiFi se pueden utilizar diferentes modelos de propagación dependiendo las características de la red que se pretenda simular [23].
Los modelos de propagación son los siguientes:

- Friis Propagation Loss Model (friisPropagationLossModel)
- Two Ray Ground Propagation Loss Model (twoRayGroundPropagationLossModel)
- Log Distance Propagation Loss Model (logDistancePropagationLossModel)
- Path Loss Model (pathLoss)
- Log-Normal Shadowing Propagation Loss Model (logNormalShadowingPropagation-LossModel)
- ITU Propagation Loss Model(ITUPropagationLossModel)
- Young Propagation Loss Model (youngModel)
- Okumura Hata Propagation Loss Model (okumuraHataPropagationLossModel)
- Jakes Propagation Loss (ModeljakesPropagationLossModel)

Algunos de ellos como son Log-Distance Propagation Model y el ITU Propagation Model están implementados para espacios indoor mientras que otros están diseñados para espacios outdoor como es Two-Ray-Ground Propagation Model, ya que para distancias cortas los resultados no son buenos.

Para introducir los modelos de propagación en el script se utliza la siguiente forma:

```
net.propagationModel('friisPropagationLossModel', sL=2)
```

Donde sL es las pérdidas del sistema.

Otras características que se pueden especificar son las siguientes:

- rssi (Received Signal Strength Indicator)
- model
- exp (Exponent)
- sl (System Loss)
- lF (Floor penetration loss factor)
- pL (Power Loss Coefficient)
- nFloors (Number of floors)
- gRandom (Gaussian random variable)

4

4. Ejemplo práctico

En este capítulo se van a realizar diferentes tipos de pruebas sobre diferentes escenarios, aumentando la dificultad en cada uno de ellos. De esta forma, se pondrá en práctica todo aquello que se ha mencionado y sobre lo que se ha trabajado anteriormente.

- Prueba de escenarios simples sobre MiniEdit usando un controlador remoto.
- Prueba de escenarios a través de Python con estaciones estáticas inalámbricas utilizando un controlador remoto
- Prueba de escenarios a través de Python con estaciones móviles inalámbricas utilizando un controlador remoto

4.1. Prueba de escenarios simples sobre MiniEdit usando un controlador remoto

Se empezará por crear el escenario, para lo que se utilizará MiniEdit. Para ejecutar MiniEdit se deben seguir los siguientes pasos:

```
cd Mininet-wifi
cd Mininet
cd examples
./miniedit.py
```

Tras esto aparecerá la pantalla en blanco con todos los elementos necesarios para crear la topología del escenario. En este caso el escenario que se va a crear dispone de 8 hosts (h1-h8), 9 switches OpenFlow (s1-s9) y finalmente, un controlador (c0). El controlador deberá estar conectado a todos los switches OpenFlow, como se puede ver en la imagen (Figura 12), si fuera de otra forma no funcionaría correctamente el escenario.

Una vez se dispone de la toplogía como la de la figura anterior se procede a cambiar las características necesarias.
Lo primero es indicar que se abra la ventana del cliente, para ello,se pulsa en Edit, Preferences y se clica en start CLI.
El siguiente detalle a modificar es indicar que tipo de controlador se va a utilizar, ya que de serie aparecerá un controlador no remoto y el que se va a utilizar será remoto. Para modificarlo, se pincha sobre el controlador con el botón izquierdo y en Properties se indica Remote Controller.

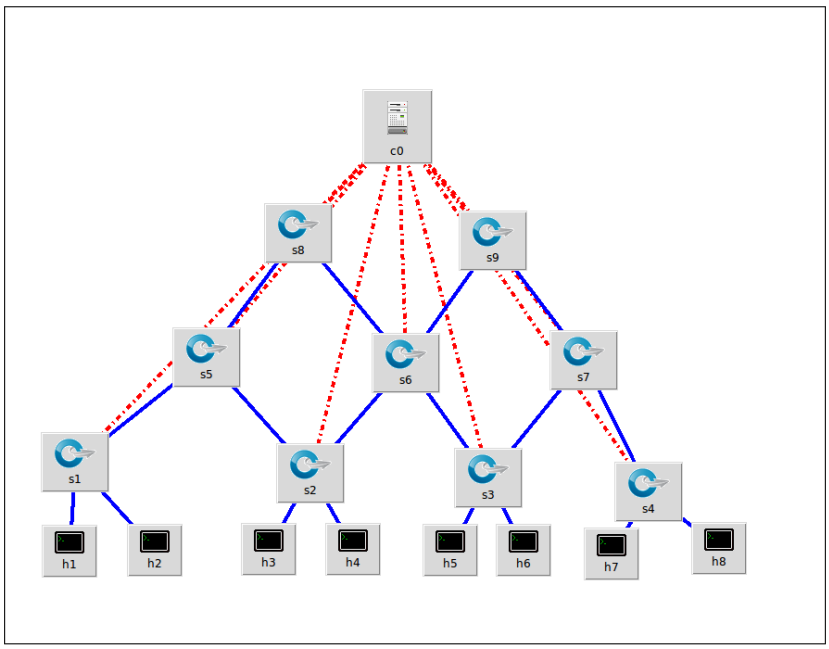


Figura 12: Topología del primer escenario

Ahora, una vez esté preparado el escenario, antes de pulsar sobre Run, se abre el controlador, en este caso el controlador que se va a emplear es ODL. Para ello, se llevarán acabo los pasos explicados en el apartado de ODL.

A continuación, se debe enviar un ping a todos los host de nuestra red, para que nuestro controlador pueda ver la toplogía del escenario y a su vez, comprobar que la topología está bien creada; ya que si ocurriera que alguno de los pings no alcanzara su destino se debería repasar todos los pasos anteriores hasta averiguar dónde está el fallo.

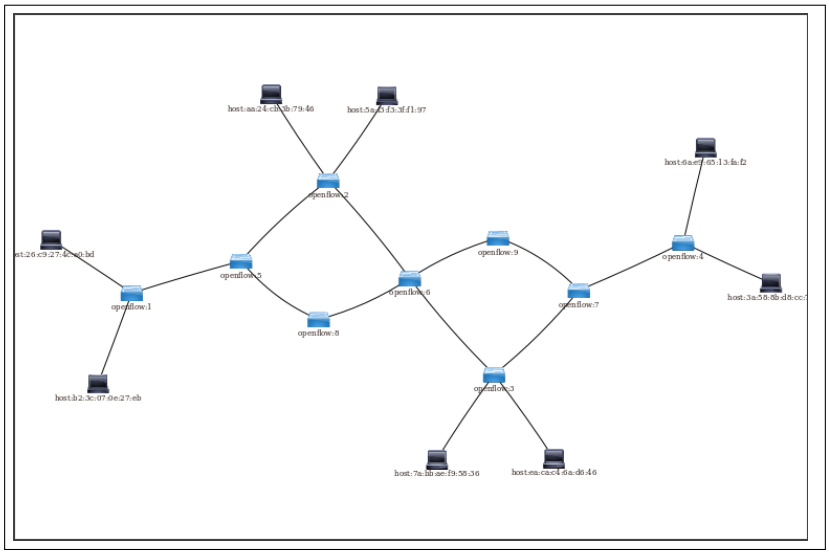


Figura 13: Topología con ODL del primer escenario

Se comprueba que todo funciona y se procede a realizar diferentes pruebas.
Lo primero que se va a hacer es confirmar las configuraciones de los switches de la red, de esta forma se podrá verificar que todo está correctamente conectado, y que cada switch esta conectado al controlador. Para ello, se pincha en la pestaña Run y después en Show OVS Summary. Con esto se obtendrá lo siguiente:

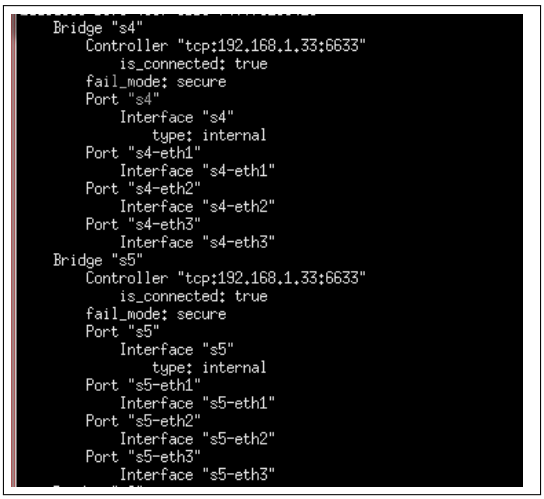


Figura 14: Sumario de conexión de los switches s4 y s5

Tras la prueba realizada se realizará la comprobación de las tablas de flujo de cada uno de los switches. Para ésto, es necesario abrir una terminal del ordenador,es decir, no se puede usar una terminal correspondiente a uno de los hosts de nuestro escenario. La forma para abrir una terminal desde la interfaz de MiniEdit es pulsando sobre Run y tras esto pulsar sobre Root Terminal. Una vez en esta ventana de comandos para poder ver las tablas de flujo se debe escribir el siguiente comando:

```
sudo ovs-ofctl dump-flows s1
```

Con este comando se obtendrá la tabla de flujos para s1, en el caso del escenario que se emplea en el ejemplo habrá que ir sustituyendo s1 por s2,s3,s4,s5,s6,s7,s8 y s9. El resultado obtenido será el siguiente:

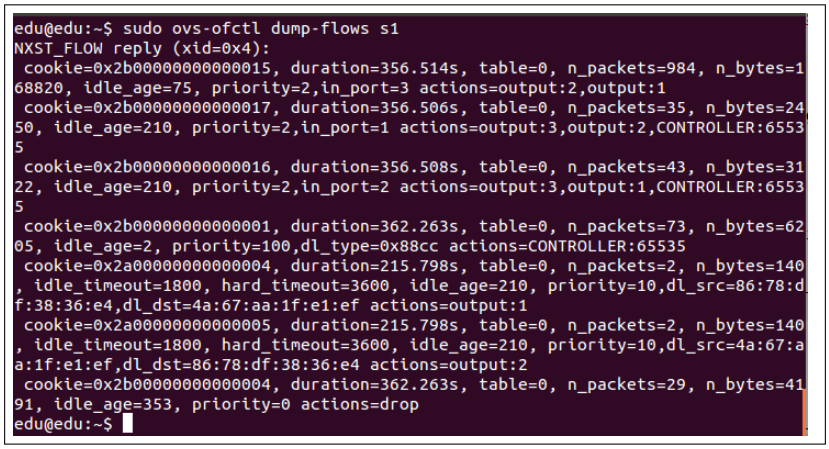


Figura 15: Tabla de flujo para el switch s1

```
edu@edu:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x2b00000000000001, duration=245.544s, table=0, n_packets=359, n_bytes=6
2065, idle_age=28, priority=2,in_port=3 actions=output:1,output:2,output:4
 cookie=0x2b00000000000000, duration=245.544s, table=0, n_packets=37, n_bytes=25
90, idle_age=99, priority=2,in_port=1 actions=output:3,output:2,output:4,CONTROL
LER:65535
 cookie=0x2b00000000000003, duration=245.544s, table=0, n_packets=618, n_bytes=1
05511, idle_age=28, priority=2,in_port=4 actions=output:1,output:3,output:2
 cookie=0x2b00000000000002, duration=245.544s, table=0, n_packets=44, n_bytes=32
20, idle_age=99, priority=2,in_port=2 actions=output:1,output:3,output:4,CONTROL
LER:65535
 cookie=0x2b00000000000003, duration=251.294s, table=0, n_packets=102, n_bytes=8
670, idle_age=1, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2a00000000000006, duration=104.783s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, idle_age=99, priority=10,dl_src=92:33:bd
:ae:5e:0d,dl_dst=f2:71:05:e8:1d:c4 actions=output:1
 cookie=0x2a00000000000007, duration=104.783s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, idle_age=99, priority=10,dl_src=f2:71:05
:e8:1d:c4,dl_dst=92:33:bd:ae:5e:0d actions=output:2
 cookie=0x2b00000000000005, duration=251.34s, table=0, n_packets=53, n_bytes=842
2, idle_age=242, priority=0 actions=drop
edu@edu:~$
```

Figura 16: Tabla de flujo para el switch s2

Ahora se va a observar qué pasaría en caso de que uno de los enlaces de la red se cayera.
Lo primero es realizar un ping entre h1 y h3.

```
*** Starting CLI:
mininet-wifi> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.11 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.492 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.477 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.486 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.455 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.415 ms
```

Figura 17: Ping entre h1 y h3

Como se puede observar en la imagen (Figura 17), no hay ningun problema y el ping alcanza su objetivo.
Ahora se va a realizar la simulación de un enlace caído, para ello, se selecciona el enlace entre s5 y s2 con el botón izquierdo y se pulsa Link Down.

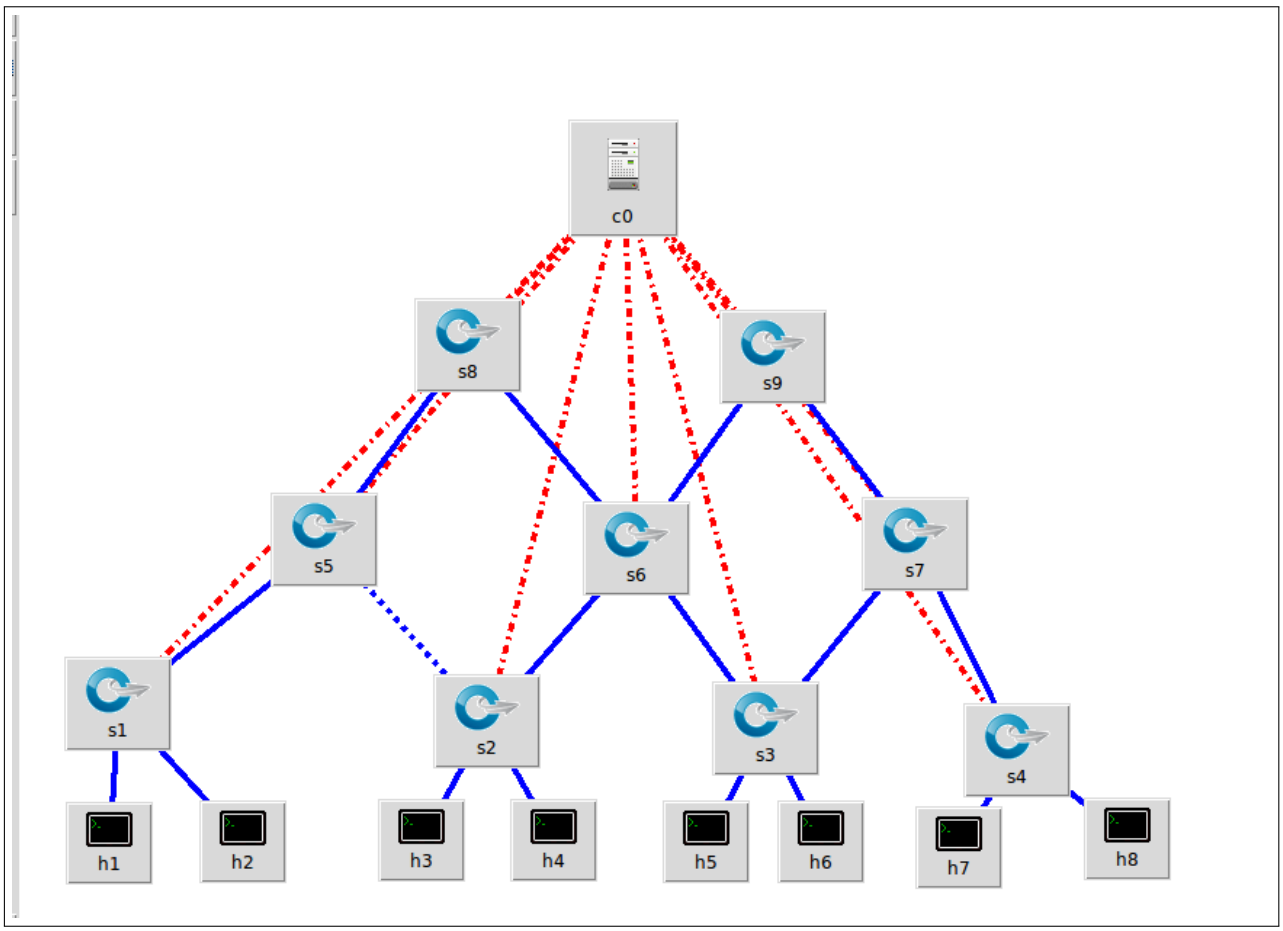


Figura 18: Simulación de un enlace caído

Tras ésto, se repite el ping, y se observa que sucede. Como se puede ver, ahora tarda más en enviar el primer ping, esto se debe a que está estableciendo un nuevo camino, pero finalmente encuentra un nuevo camino y envía el ping.

```
mininet-wifi> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=27 ttl=64 time=0.606 ms
64 bytes from 10.0.0.3: icmp_seq=28 ttl=64 time=0.265 ms
64 bytes from 10.0.0.3: icmp_seq=29 ttl=64 time=0.366 ms
64 bytes from 10.0.0.3: icmp_seq=30 ttl=64 time=0.446 ms
64 bytes from 10.0.0.3: icmp_seq=31 ttl=64 time=0.469 ms
64 bytes from 10.0.0.3: icmp_seq=32 ttl=64 time=0.358 ms
64 bytes from 10.0.0.3: icmp_seq=33 ttl=64 time=0.482 ms
64 bytes from 10.0.0.3: icmp_seq=34 ttl=64 time=0.458 ms
64 bytes from 10.0.0.3: icmp_seq=35 ttl=64 time=0.429 ms
```

Figura 19: Ping entre h1 y h3 con enlace caído

Ahora se va obtener el equivalente del escenario que se ha creado a través de MiniEdit a un script del nivel 2. Para ello basta con pulsar sobre `File` y después en `Export Level 2 Script` el resultado que se obtendrá será el obtenido en la figura 20 y en la figura 21.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='192.168.1.37',
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s9 = net.addSwitch('s9', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h6 = net.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
```

Figura 20: Script del escenario de la prueba1(Parte1)

```
    h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(h1, s1)
    net.addLink(s1, h2)
    net.addLink(h3, s2)
    net.addLink(s2, h4)
    net.addLink(h5, s3)
    net.addLink(s3, h6)
    net.addLink(s1, s5)
    net.addLink(s5, s2)
    net.addLink(s2, s6)
    net.addLink(s6, s3)
    net.addLink(s3, s7)
    net.addLink(s7, s4)
    net.addLink(s4, h7)
    net.addLink(s4, h8)
    net.addLink(s9, s6)
    net.addLink(s9, s7)
    net.addLink(s5, s8)
    net.addLink(s8, s6)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s5').start([c0])
    net.get('s3').start([c0])
    net.get('s2').start([c0])
    net.get('s9').start([c0])
    net.get('s8').start([c0])
    net.get('s4').start([c0])
    net.get('s7').start([c0])
    net.get('s6').start([c0])
    net.get('s1').start([c0])

    info( '*** Post configure switches and hosts\n')
```

Figura 21: Script del escenario de la prueba1(Parte2)

Se puede observar como todos los elementos que fueron creados a través de la interfaz de MiniEdit se encuentran en el script que se ha obtenido.

Como último ejercicio sobre este escenario, se van a realizar una serie de capturas con Wireshark para observar el comportamiento de este escenario a otro nivel, observando el tráfico OpenFlow y el tráfico de datos.
El momento en el que se realizará la captura será durante un ping entre h1 y h3, pero para ello hay que saber la interfaz que se va a seleccionar en Wireshark, para saber esta interfaz se utilizará el controlador ODL, como se puede ver en la siguiente imagen (Figura 22):

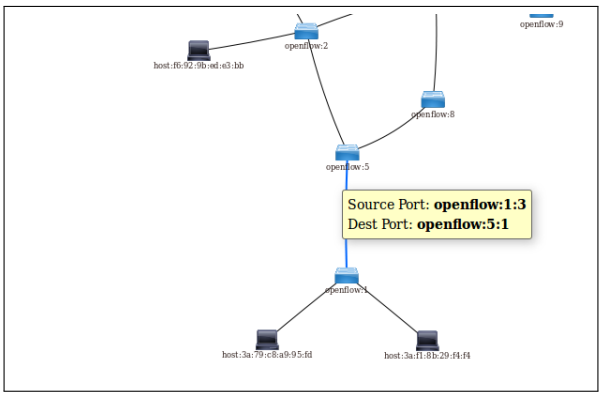
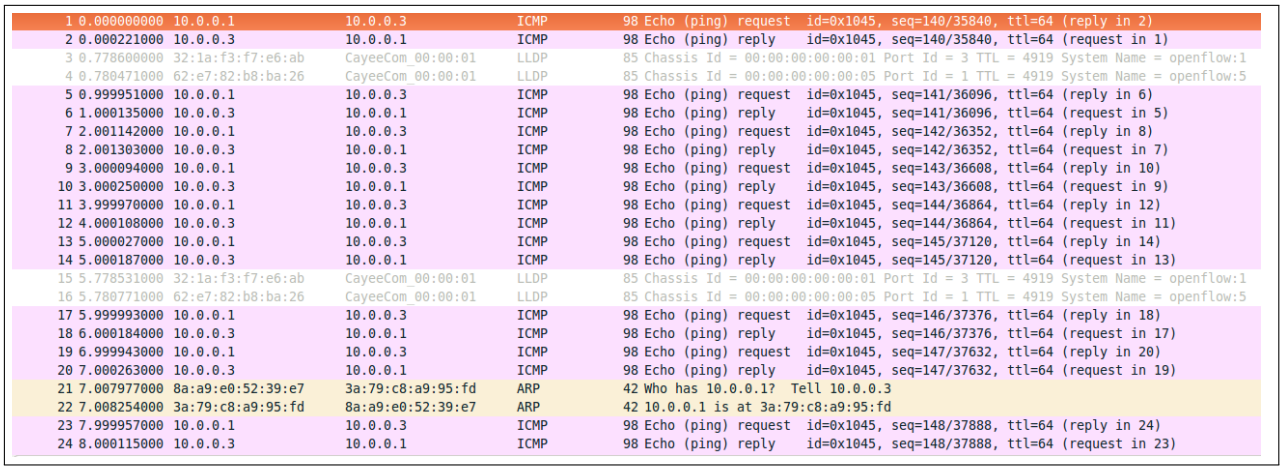


Figura 22: Selección de interfaz para la captura de tráfico en Wireshark

En la imagen se puede ver que la interfaz en la que se va a capturar es la interfaz s5-eth1.
Se pone Wireshark a capturar y se obtendrá lo siguiente:



1	0.000000000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=140/35840, ttl=64 (reply in 2)
2	0.000221000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=140/35840, ttl=64 (request in 1)
3	0.778600000	32:1a:f3:f7:e6:ab	CayeeCom_00:00:01	LLDP	85 Chassis Id = 00:00:00:00:00:01 Port Id = 3 TTL = 4919 System Name = openflow:1
4	0.780471000	62:e7:82:b8:ba:26	CayeeCom_00:00:01	LLDP	85 Chassis Id = 00:00:00:00:00:05 Port Id = 1 TTL = 4919 System Name = openflow:5
5	0.999951000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=141/36096, ttl=64 (reply in 6)
6	1.000135000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=141/36096, ttl=64 (request in 5)
7	2.001142000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=142/36352, ttl=64 (reply in 8)
8	2.001303000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=142/36352, ttl=64 (request in 7)
9	3.000094000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=143/36608, ttl=64 (reply in 10)
10	3.000250000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=143/36608, ttl=64 (request in 9)
11	3.999970000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=144/36864, ttl=64 (reply in 12)
12	4.000108000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=144/36864, ttl=64 (request in 11)
13	5.000027000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=145/37120, ttl=64 (reply in 14)
14	5.000187000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=145/37120, ttl=64 (request in 13)
15	5.778531000	32:1a:f3:f7:e6:ab	CayeeCom_00:00:01	LLDP	85 Chassis Id = 00:00:00:00:00:01 Port Id = 3 TTL = 4919 System Name = openflow:1
16	5.780771000	62:e7:82:b8:ba:26	CayeeCom_00:00:01	LLDP	85 Chassis Id = 00:00:00:00:00:05 Port Id = 1 TTL = 4919 System Name = openflow:5
17	5.999993000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=146/37376, ttl=64 (reply in 18)
18	6.000184000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=146/37376, ttl=64 (request in 17)
19	6.999943000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=147/37632, ttl=64 (reply in 20)
20	7.000263000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=147/37632, ttl=64 (request in 19)
21	7.007977000	8a:a9:e0:52:39:e7	3a:79:c8:a9:95:fd	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3
22	7.008254000	3a:79:c8:a9:95:fd	8a:a9:e0:52:39:e7	ARP	42 10.0.0.1 is at 3a:79:c8:a9:95:fd
23	7.999957000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x1045, seq=148/37888, ttl=64 (reply in 24)
24	8.000115000	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply id=0x1045, seq=148/37888, ttl=64 (request in 23)

Figura 23: Captura de tráfico en la interfaz s5-eth1 del escenario prueba1

Se puede observar que capturando en esta interfaz se obtienen paquetes de tres tipos de protocolos ICMP, LLDP y ARP, si lo que se desea es obtener paquetes del protocolo OpenFlow habrá que capturar en otra interfaz como se hará a continuación.
En primer lugar en la imagen se puede observar el ping desde h1 a h3. En segundo lugar, se encuentra el protocolo LLDP cuya fuente es el switch OpenFlow s1, para saber esta información, se ha usado el controlador como se puede ver en la siguiente imagen (Figura 24):

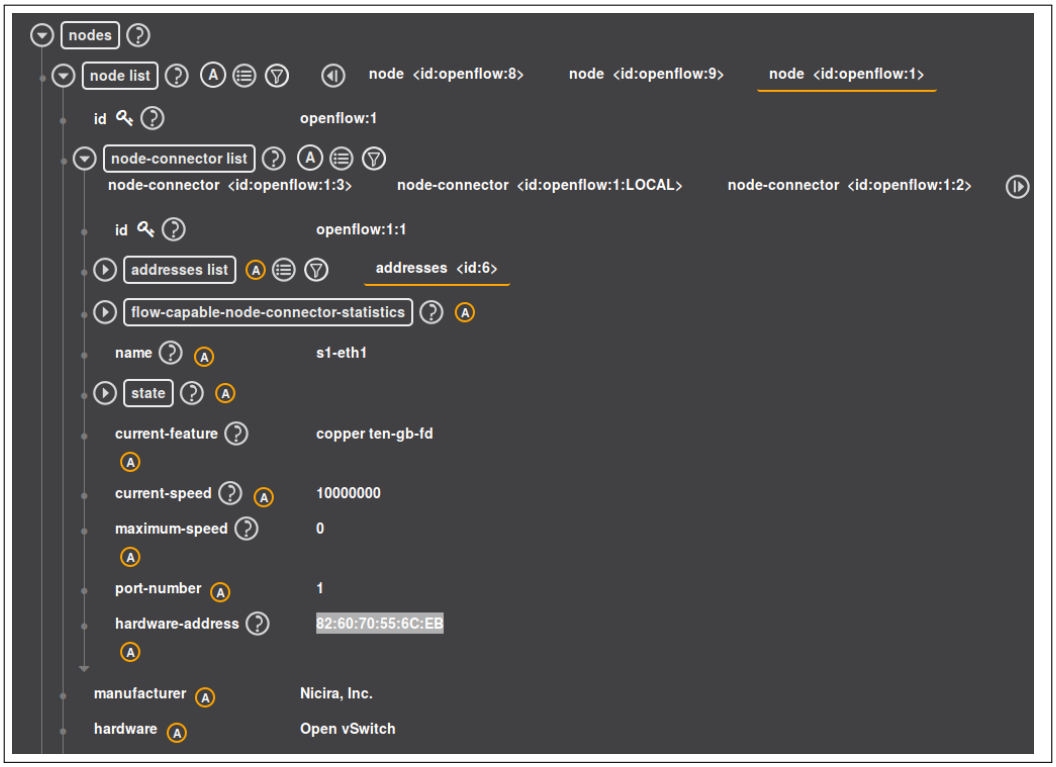
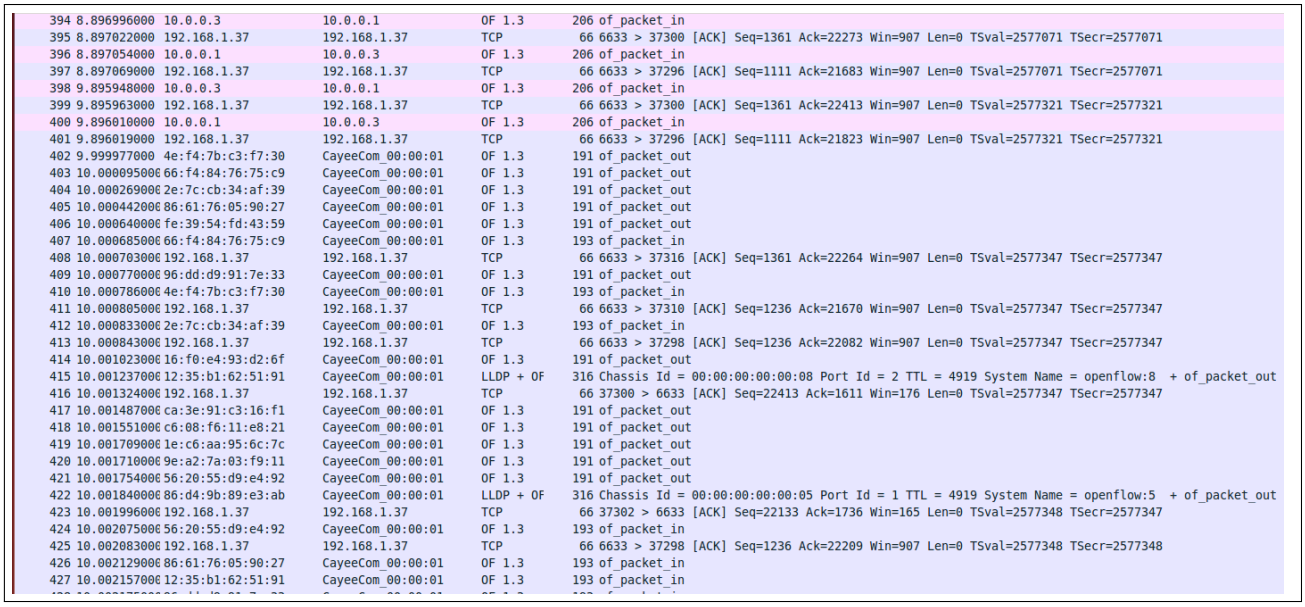


Figura 24: Descubriendo la dirección mac del switch OpenFLow s1

Para llegar hasta lo obtenido en la imagen, se entrará en Yang UI, una vez aquí se entrará en OpenDayLight-inventory, se desplegará, a su vez se desplegará Operationals y se pulsará sobre Nodes, y una vez aquí, se pulsará sobre Send y se conseguirá toda la información relacionada con los nodos de la red.

Por último en esta captura aparece el protocolo ARP, utilizado para obtener la dirección MAC correspondiente a cada dirección IP.

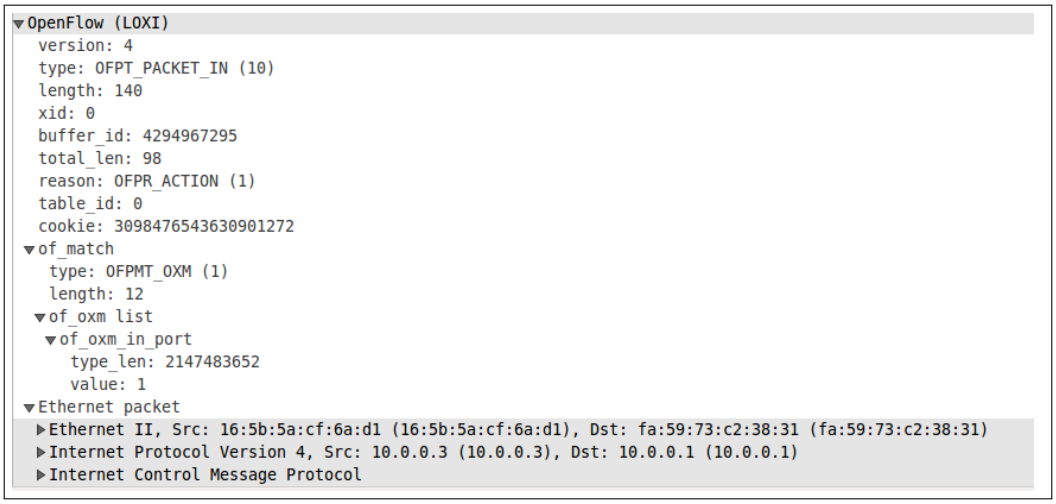
La captura que se va a analizar a continuación se realiza en la interfaz de loopback:Lo donde se pueden observar las peticiones de estadísticas y el protocolo OpenFlow, como se puede ver en la siguiente captura (Figura 25):



394	8.896996000	10.0.0.3	10.0.0.1	OF 1.3	206	of_packet_in
395	8.897022000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37300 [ACK] Seq=1361 Ack=22273 Win=907 Len=0 TSval=2577071 TSecr=2577071
396	8.897054000	10.0.0.1	10.0.0.3	OF 1.3	206	of_packet_in
397	8.897069000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37296 [ACK] Seq=1111 Ack=21683 Win=907 Len=0 TSval=2577071 TSecr=2577071
398	9.895948000	10.0.0.3	10.0.0.1	OF 1.3	206	of_packet_in
399	9.895963000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37300 [ACK] Seq=1361 Ack=22413 Win=907 Len=0 TSval=2577321 TSecr=2577321
400	9.896010000	10.0.0.1	10.0.0.3	OF 1.3	206	of_packet_in
401	9.896019000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37296 [ACK] Seq=1111 Ack=21823 Win=907 Len=0 TSval=2577321 TSecr=2577321
402	9.999977000	4e:f4:7b:c3:f7:30	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
403	10.000095000	66:f4:84:76:75:c9	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
404	10.000269000	2e:7c:cb:34:af:39	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
405	10.000442000	86:61:76:05:90:27	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
406	10.000640000	fe:39:54:fd:43:59	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
407	10.000685000	66:f4:84:76:75:c9	CayeeCom_00:00:01	OF 1.3	193	of_packet_in
408	10.000703000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37316 [ACK] Seq=1361 Ack=22264 Win=907 Len=0 TSval=2577347 TSecr=2577347
409	10.000770000	96:dd:d9:91:7e:33	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
410	10.000786000	4e:f4:7b:c3:f7:30	CayeeCom_00:00:01	OF 1.3	193	of_packet_in
411	10.000805000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37310 [ACK] Seq=1236 Ack=21670 Win=907 Len=0 TSval=2577347 TSecr=2577347
412	10.000833000	2e:7c:cb:34:af:39	CayeeCom_00:00:01	OF 1.3	193	of_packet_in
413	10.000843000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37298 [ACK] Seq=1236 Ack=22082 Win=907 Len=0 TSval=2577347 TSecr=2577347
414	10.001023000	16:f0:e4:93:d2:6f	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
415	10.001237000	12:35:b1:62:51:91	CayeeCom_00:00:01	LLDP + OF	316	Chassis Id = 00:00:00:00:00:08 Port Id = 2 TTL = 4919 System Name = openflow:8 + of_packet_out
416	10.001324000	192.168.1.37	192.168.1.37	TCP	66	37300 > 6633 [ACK] Seq=22413 Ack=1611 Win=176 Len=0 TSval=2577347 TSecr=2577347
417	10.001487000	ca:3e:91:c3:16:f1	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
418	10.001551000	c6:08:f6:11:e8:21	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
419	10.001709000	1e:c6:aa:95:6c:7c	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
420	10.001710000	9e:a2:7a:03:f9:11	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
421	10.001754000	56:20:55:d9:e4:92	CayeeCom_00:00:01	OF 1.3	191	of_packet_out
422	10.001840000	86:d4:9b:89:e3:ab	CayeeCom_00:00:01	LLDP + OF	316	Chassis Id = 00:00:00:00:00:05 Port Id = 1 TTL = 4919 System Name = openflow:5 + of_packet_out
423	10.001996000	192.168.1.37	192.168.1.37	TCP	66	37302 > 6633 [ACK] Seq=22133 Ack=1736 Win=165 Len=0 TSval=2577348 TSecr=2577347
424	10.002075000	56:20:55:d9:e4:92	CayeeCom_00:00:01	OF 1.3	193	of_packet_in
425	10.002083000	192.168.1.37	192.168.1.37	TCP	66	6633 > 37298 [ACK] Seq=1236 Ack=22209 Win=907 Len=0 TSval=2577348 TSecr=2577348
426	10.002129000	86:61:76:05:90:27	CayeeCom_00:00:01	OF 1.3	193	of_packet_in
427	10.002157000	12:35:b1:62:51:91	CayeeCom_00:00:01	OF 1.3	193	of_packet_in

Figura 25: Captura de tráfico OpenFlow en la interfaz de Loopback del escenario prueba1

En la captura aparece en primer lugar un Of_packet_in, se trata del mensaje OpenFlow con las características siguientes:



```
▼OpenFlow (LOXI)
  version: 4
  type: OFPT_PACKET_IN (10)
  length: 140
  xid: 0
  buffer_id: 4294967295
  total_len: 98
  reason: OFPR_ACTION (1)
  table_id: 0
  cookie: 3098476543630901272
 ▼of_match
   type: OFPMT_OXM (1)
   length: 12
  ▼of_oxm list
   ▼of_oxm_in_port
     type_len: 2147483652
     value: 1
 ▼Ethernet packet
  ▶Ethernet II, Src: 16:5b:5a:cf:6a:d1 (16:5b:5a:cf:6a:d1), Dst: fa:59:73:c2:38:31 (fa:59:73:c2:38:31)
  ▶Internet Protocol Version 4, Src: 10.0.0.3 (10.0.0.3), Dst: 10.0.0.1 (10.0.0.1)
  ▶Internet Control Message Protocol
```

Figura 26: Caracteristicas del mensaje OpenFlow

Dentro de este mensaje también se puede ver la posición que ocupa el protocolo OpenFlow en la encapsulación, como se puede observar en la imagen siguiente (Figura 27):

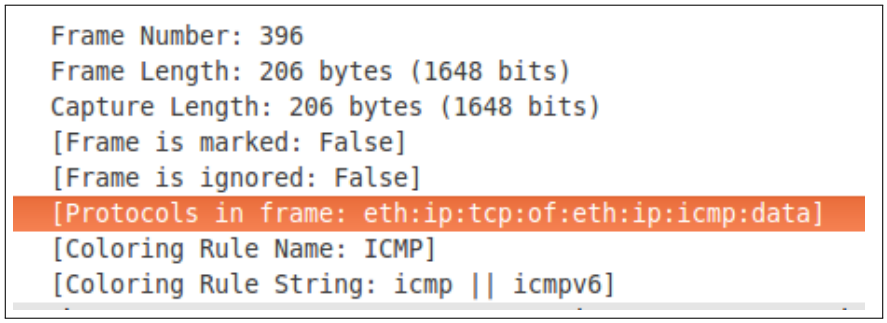


Figura 27: Encapsulación del mensaje OpenFlow

En la captura también se encuentran diferentes paquetes OpenFlow relacionados con estadísticas de puerto, de cola y de flujo, estos mensajes aparecen de dos en dos (reply y request).
Otra característica que también se puede ver en la captura es que el protocolo OpenFlow que se emplea es OpenFlow 1.3.

4.2. Prueba de escenarios a través de Python con estaciones fijas inalámbricas utilizando un controlador remoto

La primera parte de esta prueba requiere de la realización del escenario, que conlleva una serie de pasos para su implementación; para la que se utilizará un script de Python.

El primer paso consiste en importar todas aquellas clases que se necesitan:

```
#!/usr/bin/python


from mininet.net import Mininet
from mininet.node import OVSSwitch,OVSKernelSwitch, Controller, RemoteController
from mininet.topolib import TreeTopo
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.link import TCLink
```

Figura 28: Clases del segundo escenario de prueba

El siguiente paso es la creación de la topología, es decir de los puntos de acceso y de las estaciones, como a su vez, los enlaces entre ellos y con el controlador:

```
def topology():
    "Create a network."
    net = Mininet( controller=RemoteController, link=TCLink, switch=OVSKernelSwitch )
    print "*** Creating nodes"
    sta1 = net.addStation( 'sta1', ip="10.0.0.1/24" )
    sta2 = net.addStation( 'sta2', ip="10.0.0.2/24" )
    sta3 = net.addStation( 'sta3', ip="10.0.0.3/24" )
    sta4 = net.addStation( 'sta4', ip="10.0.0.4/24" )
    sta5 = net.addStation( 'sta5', ip="10.0.0.5/24" )
    sta6 = net.addStation( 'sta6', ip="10.0.0.6/24" )

    ap1 = net.addBaseStation( 'ap1', ssid="ssid_1", mode="g", channel="1" )
    ap2 = net.addBaseStation( 'ap2', ssid="ssid_2", mode="b", channel="6" )
    ap3 = net.addBaseStation( 'ap3', ssid="ssid_3", mode="g", channel="1" )
    c0 = net.addController('c0', controller=RemoteController, ip='192.168.1.37', port=6653 )

    print "*** Adding Link"
    net.addLink(ap1, ap2) #wired connection
    net.addLink(ap2, ap3)
    net.addLink(sta1, ap1)
    net.addLink(sta2, ap1)
    net.addLink(sta3, ap2)
    net.addLink(sta4, ap2)
    net.addLink(sta5, ap3)
    net.addLink(sta6, ap3)

    print "*** Starting network"
    net.build()
    c0.start()
    ap1.start( [c0] )
    ap2.start( [c0] )
    ap3.start( [c0] )
```

Figura 29: Creación de la topología del segundo escenario de prueba

Una vez se dispone del script, se pasa a comprobar si está correctamente creado. Para ello, se enviará un ping a todas las estaciones y usando la interfaz gráfica del controlador, que permite visualizar la topología se comprobará si es posible alcanzar todos los nodos de la red. En este caso, como se puede observar en la siguiente captura (Figura 30),es posible alcanzar todos los nodos ya que aparecen todos los nodos en la interfaz del controlador.

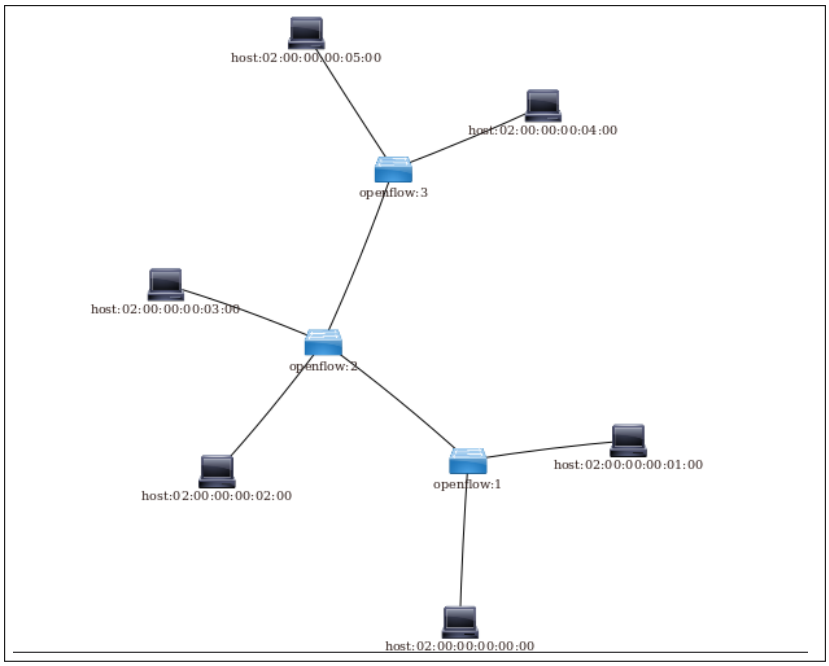


Figura 30: Visualización mediante OpenDayLight de la topología del segundo escenario

Otra forma de verificar las conexiones del diseño de la red es mediante el comando:

```
net
```

Se obtendrá:

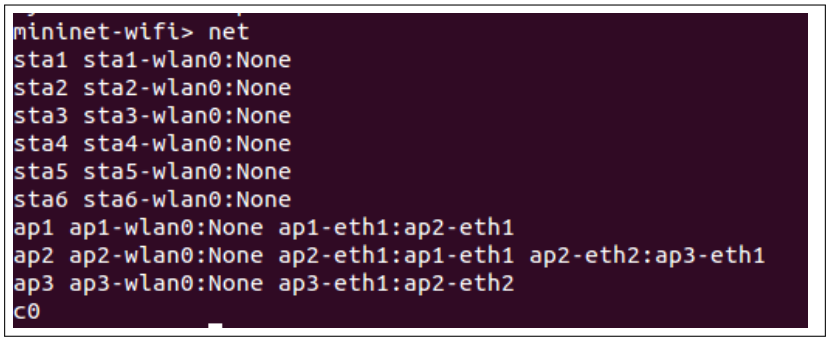


Figura 31: Conexiones de red del segundo escenario de prueba

Con ésto solo se puede verificar las conexiones de los puntos de acceso, ya que como las estaciones están conectadas de forma inalámbrica será necesario utilizar otro comando para saber que puntos de acceso son capaces de ver cada estación:

```
sta1 iw dev sta1-wlan0 link
```

Con ésto se verá la información que se desea pero solo de la sta1 por lo que se deberá ir cambiando por las diferentes estaciones.Con esto se obtendrá:

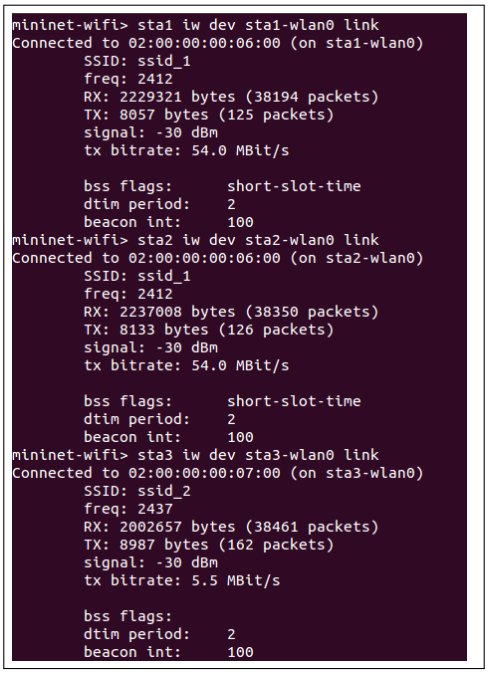


Figura 32: Verificación de las conexiones de las estaciones del escenario(Parte 1)

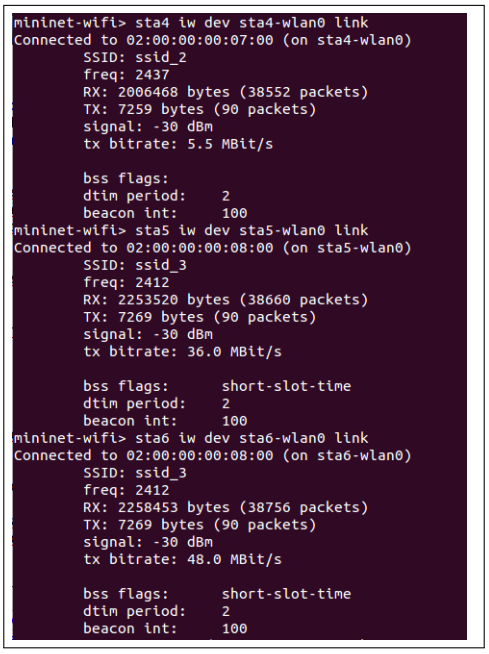


Figura 33: Verificación de las conexiones de las estaciones del escenario(Parte 2)

Ahora se va a proceder a obtener las tablas de flujo de los diferentes nodos de la red. Para ello, se utilizará el siguiente comando:

```
dpctl dump-flows
```

Con el anterior comando se obtendrá la tabla de flujos de todos los puntos de acceso de la red, es decir de ap1, ap2 y ap3; como se puede ver en la siguiente imagen(Figura 34):

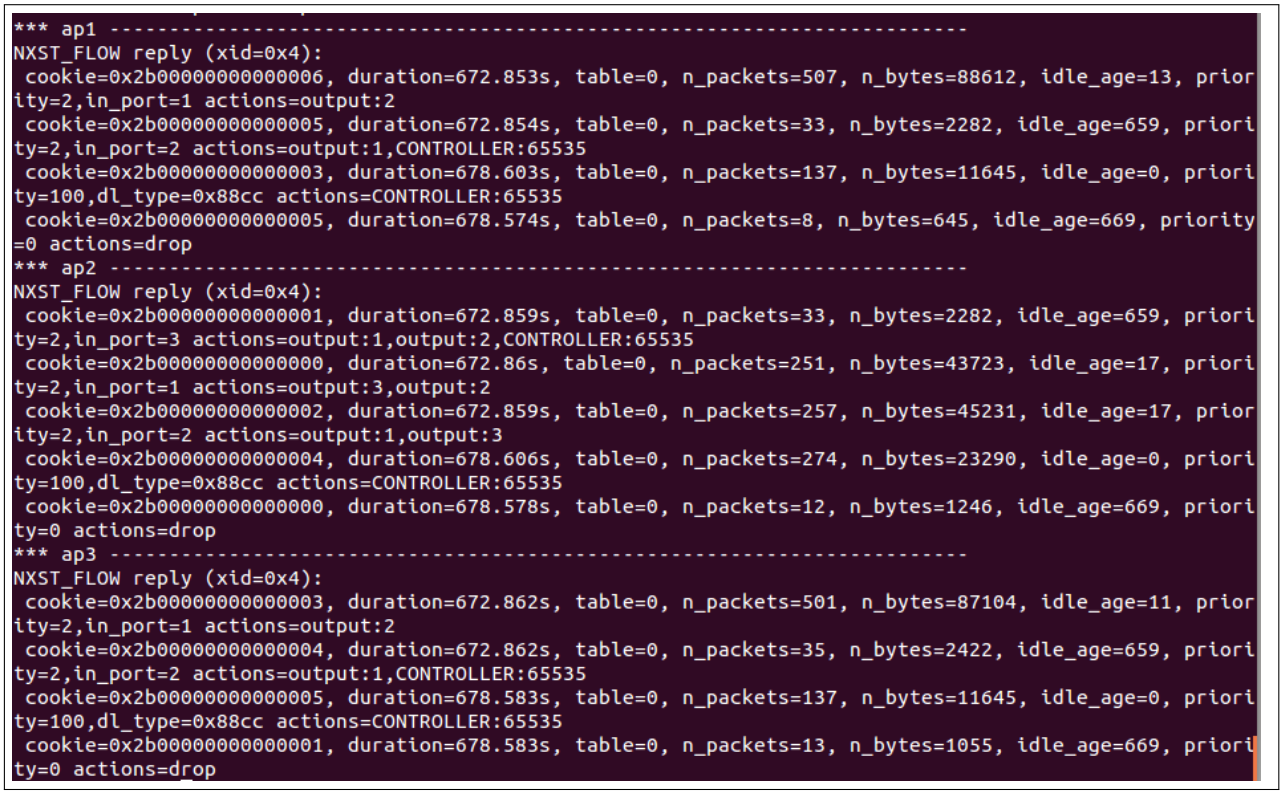


Figura 34: Tabla de flujos del escenario Prueba2

Por último se realizarán capturas en Wireshark para observar el comportamiento a otro nivel.
La captura será tomada en un ping entre sta1 y sta6. Y será analizada en diferentes interfaces, en primer lugar, en la interfaz de Loopback, tras ésta se analizará en la interfaz Ap1-wlan0, y por último en la interfaz hwsim0 ya que se trata de la interfaz de software creada por Mininet-WiFi que copia todo el tráfico wireless a todas las interfaces inalámbricas virtuales en el escenario. Ésta interfaz hwsim0 es la forma más fácil de monitorizar los paquetes inalámbricos en MIninet-WiFi.

Se empezará por la captura de tráfico en la interfaz de Loopback:

1	0.000000000	10.0.0.1	10.0.0.6	OF 1.0	182 of_packet_in
2	0.000032000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49670 [ACK] Seq=1 Ack=117 Win=907 Len=0 TSval=5125261 TSecr=5125261
3	0.000097000	10.0.0.6	10.0.0.1	OF 1.0	182 of_packet_in
4	0.000118000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49674 [ACK] Seq=1 Ack=117 Win=907 Len=0 TSval=5125261 TSecr=5125261
5	1.000769000	10.0.0.1	10.0.0.6	OF 1.0	182 of_packet_in
6	1.000782000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49670 [ACK] Seq=1 Ack=233 Win=907 Len=0 TSval=5125511 TSecr=5125511
7	1.000842000	10.0.0.6	10.0.0.1	OF 1.0	182 of_packet_in
8	1.000850000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49674 [ACK] Seq=1 Ack=233 Win=907 Len=0 TSval=5125511 TSecr=5125511
9	1.999758000	10.0.0.1	10.0.0.6	OF 1.0	182 of_packet_in
10	1.999772000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49670 [ACK] Seq=1 Ack=349 Win=907 Len=0 TSval=5125761 TSecr=5125761
11	1.999828000	10.0.0.6	10.0.0.1	OF 1.0	182 of_packet_in
12	1.999836000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49674 [ACK] Seq=1 Ack=349 Win=907 Len=0 TSval=5125761 TSecr=5125761
13	2.010775000	02:00:00:00:05:00	02:00:00:00:00:00	OF 1.0	126 of_packet_in
14	2.010788000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49674 [ACK] Seq=1 Ack=409 Win=907 Len=0 TSval=5125764 TSecr=5125764
15	2.010986000	02:00:00:00:00:00	02:00:00:00:05:00	OF 1.0	126 of_packet_in
16	2.010996000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49670 [ACK] Seq=1 Ack=409 Win=907 Len=0 TSval=5125764 TSecr=5125764
17	2.161817000	192.168.1.37	192.168.1.37	OF 1.0	122 of_flow_stats_request
18	2.162109000	192.168.1.37	192.168.1.37	OF 1.0	462 of_flow_stats_reply
19	2.162126000	192.168.1.37	192.168.1.37	TCP	66 openflow > 49674 [ACK] Seq=57 Ack=805 Win=907 Len=0 TSval=5125801 TSecr=5125801
20	2.167291000	192.168.1.37	192.168.1.37	OF 1.0	86 of_port_stats_request
21	2.167669000	192.168.1.37	192.168.1.37	OF 1.0	390 of_port_stats_reply
22	2.169590000	192.168.1.37	192.168.1.37	OF 1.0	86 of_queue_stats_request
23	2.169699000	192.168.1.37	192.168.1.37	OF 1.0	78 of_queue_stats_reply
24	2.171167000	192.168.1.37	192.168.1.37	OF 1.0	78 of_table_stats_request
25	2.171413000	192.168.1.37	192.168.1.37	OF 1.0	16334 of_table_stats_reply

Figura 35: Captura de tráfico en la interfaz de Loopback del escenario prueba2

Como se puede observar en la imagen (Figura 35) la única diferencia respecto a la captura del apartado anterior es la utilización de una versión diferente del protocolo OpenFlow, en este caso, se utiliza el protocolo OpenFlow 1.0.
En la figura que aparece a continuación (Figura 36) se puede ver el paquete OpenFlow, donde se puede resaltar lo destacado anteriormente:

```
▼ OpenFlow (LOXI)
    version: 1
    type: OFPT_PACKET_IN (10)
    length: 116
    xid: 0
    buffer_id: 4294967295
    total_len: 98
    in_port: 2
    reason: OFPR_ACTION (1)
  ▼ Ethernet packet
    ▶ Ethernet II, Src: 02:00:00:00:00:00 (02:00:00:00:00:00), Dst: 02:00:00:00:05:00 (02:00:00:00:05:00)
    ▶ Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.6 (10.0.0.6)
    ▶ Internet Control Message Protocol
```

Figura 36: Captura de tráfico en la interfaz de Loopback del escenario prueba2

Ahora se realiza la captura en la interfaz Ap1-wlan0:

1	0.000000000	10.0.0.1	10.0.0.6	ICMP	98 Echo (ping) request id=0x64e8, seq=175/44800, ttl=64 (reply in 2)
2	0.000494000	10.0.0.6	10.0.0.1	ICMP	98 Echo (ping) reply id=0x64e8, seq=175/44800, ttl=64 (request in 1)
3	0.999948000	10.0.0.1	10.0.0.6	ICMP	98 Echo (ping) request id=0x64e8, seq=176/45056, ttl=64 (reply in 4)
4	1.000369000	10.0.0.6	10.0.0.1	ICMP	98 Echo (ping) reply id=0x64e8, seq=176/45056, ttl=64 (request in 3)
5	1.999906000	10.0.0.1	10.0.0.6	ICMP	98 Echo (ping) request id=0x64e8, seq=177/45312, ttl=64 (reply in 6)
6	2.000373000	10.0.0.6	10.0.0.1	ICMP	98 Echo (ping) reply id=0x64e8, seq=177/45312, ttl=64 (request in 5)
7	2.178991000	02:00:00:00:06:00	CayeeCom_00:00:01	LLDP	85 Chassis Id = 00:00:00:00:00:01 Port Id = 2 TTL = 4919 System Name = openflow:1
8	2.999985000	10.0.0.1	10.0.0.6	ICMP	98 Echo (ping) request id=0x64e8, seq=178/45568, ttl=64 (reply in 9)
9	3.000428000	10.0.0.6	10.0.0.1	ICMP	98 Echo (ping) reply id=0x64e8, seq=178/45568, ttl=64 (request in 8)
10	3.008367000	02:00:00:00:05:00	02:00:00:00:00:00	ARP	42 Who has 10.0.0.1? Tell 10.0.0.6
11	3.008434000	02:00:00:00:00:00	02:00:00:00:05:00	ARP	42 10.0.0.1 is at 02:00:00:00:00:00

Figura 37: Captura de tráfico en la interfaz Ap1-wlan0 del escenario prueba2

En la figura 37 se aprecia que el resultado obtenido es similar al obtenido en el apartado anterior cuando se captura el tráfico en una de las interfaces de los nodos, obteniendo en la captura paquetes ICMP, ARP y LLDP.

Por último, se realiza la captura en la interfaz hwsim0, pero previamente es necesario activarla como se indicó en apartados anteriores, para activarla es necesario escribir en la ventana del cliente lo siguiente:

```
sh ifconfig hwsim0 up
```

Una vez esté la interfaz activada ya se puede realizar la captura mediante Wireshark:

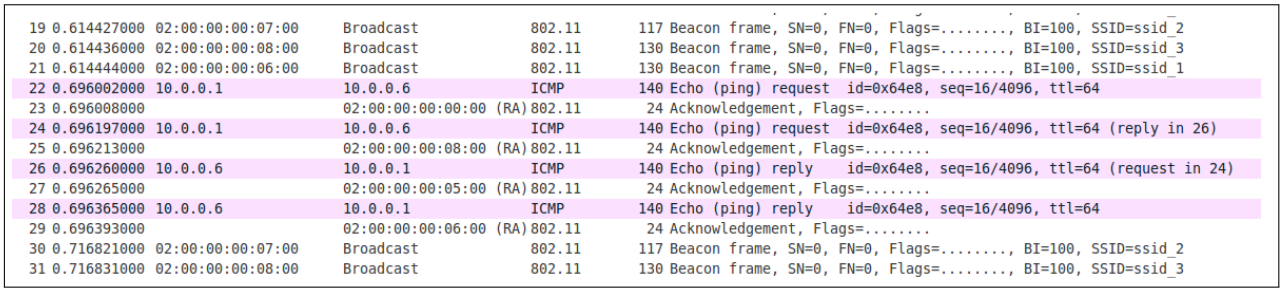


Figura 38: Captura de tráfico en la interfaz hwsim0 del escenario prueba2

En la captura de la figura hay que destacar las tramas IEEE 802.11, ya que se trata de una conexión inalámbrica la de este escenario, y mediante esta interfaz como ya se ha dicho es la mejor forma de observar este tipo de trama.

4.3. Prueba de escenarios a través de Python con estaciones móviles inalámbricas utilizando un controlador remoto

Como en la prueba anterior, lo primero que se debe crear es la topología que se va a emplear, y como en el caso anterior también se utilizará un script de Python. La primera parte es importar las clases que se van a utilizar, estas clases coinciden con las empleadas en la prueba2, como se puede observar en la imagen siguiente (Figura 39):

```
#!/usr/bin/python

"""
This example shows how to work with different APs
"""
from mininet.net import Mininet
from mininet.node import OVSKernelSwitch, Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink
```

Figura 39: Clases tercer escenario de prueba

Una vez se dispone de las clases se procede a la creación de los diferentes nodos que se van a utilizar, es decir, los puntos de acceso y las estaciones móviles, y también el tipo de movimiento que utilizarán las estaciones; se puede observar en la segunda de las imagenes (Figura 41) siguientes que el tipo de movimiento seleccionado será RandomDirection (movimiento aleatorio):

```
def topology():

    "Create a network."
    net = Mininet( controller=RemoteController, link=TCLink, switch=OVSKernelSwitch )

    print "*** Creating nodes"
    sta1 = net.addStation( 'sta1', ip='10.0.0.2/8' )
    sta2 = net.addStation( 'sta2', ip='10.0.0.3/8' )
    sta3 = net.addStation( 'sta3', ip='10.0.0.4/8' )
    sta4 = net.addStation( 'sta4', ip='10.0.0.5/8' )
    ap1 = net.addBaseStation( 'ap1', ssid= 'ssid_1', mode= 'g', channel= '1', position='50,100,0' )
    ap2 = net.addBaseStation( 'ap2', ssid= 'ssid_2', mode= 'b', channel= '1', position='150,100,0' )
    c0 = net.addController( 'c0', controller=RemoteController, ip='192.168.1.33', port=6653  )

    print "*** Associating and Creating links"
    net.addLink(ap1, ap2)
    net.addLink(sta1, ap1)
    net.addLink(sta2, ap1)
    net.addLink(sta3, ap2)
    net.addLink(sta4, ap2)
```

Figura 40: Creación topología tercer escenario de prueba

```
print "*** Starting network"
net.build()
c0.start()
ap1.start( [c0] )
ap2.start( [c0] )
"""uncomment to plot graph"""
net.plotGraph(max_x=200, max_y=200)

"""Seed"""
net.seed(20)

"*** Available models: RandomWalk, TruncatedLevyWalk, RandomDirection, RandomWayPoint, GaussMarkov, ReferencePoint, TimeVariantCommunity ***"
net.startMobility(startTime=0, model='RandomDirection', max_x=200, min_y=100, max_y=150, min_v=0.6, max_v=0.6)
```

Figura 41: Movimiento empleado en el tercer escenario de prueba

Ahora ya se dispone del script necesario para realizar las pruebas y capturas necesarias para el análisis del escenario. Se puede observar, que a diferencia de las pruebas anteriores, se abre un plot en el que se muestra nuestra topología, por lo que, aunque el controlador que se utiliza es el controlador remoto, no es necesario utilizar la interfaz gráfica de ODL para observar si el escenario ha sido creado correctamente.
Otro problema que se puede encontrar a la hora de querer ver la topología en el controlador OpenDayLight es que resulta muy difícil poder enviar un ping a todos los nodos, ya que estos se encuentran en movimiento y no siempre estarán conectados a uno de los puntos de acceso.

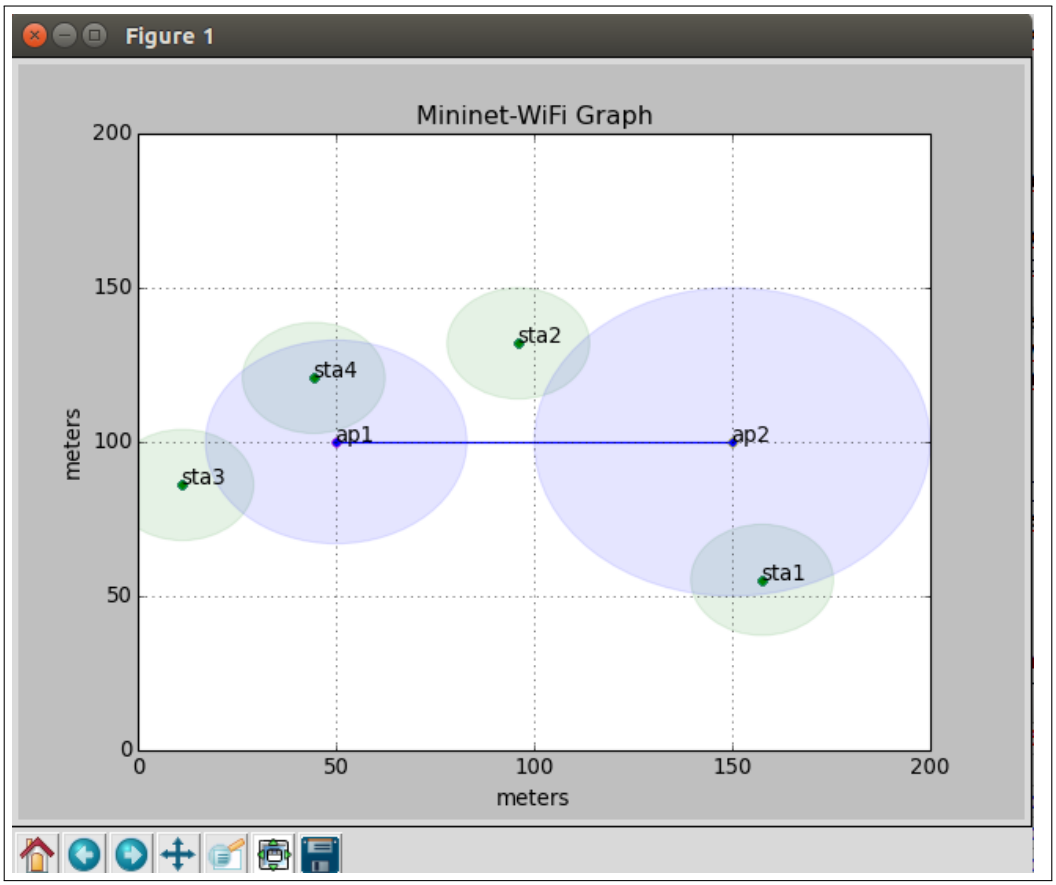


Figura 42: Visualización de la topología del tercer escenario

Para probar que es posible alcanzar las estaciones desde los diferentes puntos de acceso se envía un ping desde sta1 a sta3 en el momento en el que sta1 se sitúa sobre ap2 y sta3 se sitúa sobre ap1, ésto también se realizará entre los demás pares de estaciones.

Tras ésto se va a ver la información de cada uno de los puntos de acceso.Para hacer ésto, se utilizará el siguiente comando:

```
info ap1
```

Uno de los aspectos más importantes de Mininet-WiFi es la movilidad, por ésto, a continuación se va a hacer referencia a los diferentes tipos de movimiento y cómo funcionan. Éstos son los modelos de movilidad disponibles:

- RandomWalk
- RandomDirection
- TruncatedLevyWalk
- RandomWayPoint
- GaussMarkov
- ReferencePoint
- TimeVariantCommunity

Pero previamente a éstos, es necesario hacer referencia al modelo StochasticWalk, ya que es en el que se basan parte de los modelos mencionados.

StochasticWalk: se trata de la implementación base para los modelos con dirección uniforme elegida entre 0 y pi. Estos modelos son RandomDirection, RandomWalk y TruncatedLevyWalk.

RandomDirection: el movimiento realizado es aleatorio con una velocidad uniforme establecida entre unos valores establecidos previamente.

RandomWalk: similar al anterior pero con la diferencia de que este movimiento es más lento y el cambio de dirección es en movimientos más cortos.

TruncatedLevyWalk: se trata de un caso especial del StochasticWalk. Está basado en el documento: Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, and Song Chong. On the Levy-Walk Nature of Human Mobility. In 2008 IEEE INFOCOM - Proceedings of the 27th Conference on Computer Communications, pages 924-932. April 2008.

RandomWayPoint: los nodos se mueven alrededor de un punto de referencia como su propio nombre indica.

GaussMarkov: este modelo fue propuesto en el documento: Camp, T., Boleng, J. & Davies, V. A survey of mobility models for ad hoc network research. Wireless Communications and Mobile Computing 2, 483-502 (2002).

ReferencePoint: en esta implementación, el grupo de nodos lleva a cabo un modelo RandomDirection, mientras que los nodos siguen un camino aleatorio en torno al centro del grupo. Está basado en el documento: Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. 1999. A group mobility model for ad hoc wireless networks. In Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of

wireless and mobile systems (MSWiM '99). ACM, New York, NY, USA, 53-60.

TimeVariantCommunity: este modelo fue dispuesto en el documento: Wei-jen Hsu, Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Ahmed Helmy, "Modeling Time-variant User Mobility in Wireless Mobile Networks,ÏNFOCOM 2007, May 2007. Se trata de una variante de la definición original, de la siguiente forma: las comunidades tienen un punto de referencia donde cada miembro de esta comunidad se agrega alrededor, además, estos puntos de referencia siguen un modelo RandomDirection [24].

Tras haber estudiado los diferentes modelos de movilidad se va a observar si las diferentes estaciones son capaces de conectarse a un punto de acceso y enviar información, para posteriormente, desconectarse de este punto de acceso y conectarse a otro.
Para que analizar ésto sea mas sencillo, se va a emplear un escenario de los que vienen creados para su aprendizaje, para acceder a este escenario hay que seguir los siguientes pasos:

```
cd mininet-wifi
cd examples
gedit wifiAssociationControl.py
```

Una vez se está en la ventana del script del escenario, se van a realizar una serie de cambios para que sea más fácil su visualización. En primer lugar, se eliminarán todas las estaciones menos las estaciones sta1, sta2, sta3 y sta4, ya que sino es imposible apreciar los cambios que realizan las estaciones entre los puntos de acceso. El segundo cambio que se va a realizar es el controlador, ya que el controlador empleado por defecto no es el controlador remoto que interesa, para ello se sustituirá Controller por RemoteController.
Finalmente el código quedará de la siguiente manera:

```
def topology():
    "Create a network."
    net = Mininet( controller=RemoteController, link=TCLink, switch=OVSKernelSwitch )

    print "*** Creating nodes"
    sta1 = net.addStation( 'sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8' )
    sta2 = net.addStation( 'sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8' )
    sta3 = net.addStation( 'sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/8' )
    sta4 = net.addStation( 'sta4', mac='00:00:00:00:00:05', ip='10.0.0.5/8' )
    ap1 = net.addBaseStation( 'ap1', ssid= 'new-ssid1', mode= 'g', channel= '1', position='50,50,0' )
    ap2 = net.addBaseStation( 'ap2', ssid= 'new-ssid2', mode= 'g', channel= '1', position='70,50,0', range=30 )
    ap3 = net.addBaseStation( 'ap3', ssid= 'new-ssid3', mode= 'g', channel= '1', position='90,50,0' )
    c1 = net.addController( 'c1', controller=RemoteController, ip='192.168.1.34', port=6653  )

    print "*** Associating and Creating links"
    net.addLink(ap1, ap2)
    net.addLink(ap2, ap3)

    print "*** Starting network"
    net.build()
    c1.start()
    ap1.start( [c1] )
    ap2.start( [c1] )
    ap3.start( [c1] )

    """uncomment to plot graph"""
    net.plotGraph(max_x=120, max_y=120)

    """association control"""
    net.associationControl('ssf')

    """Seed"""
    net.seed(1)

    """ *** Available models:
                RandomWalk, TruncatedLevyWalk, RandomDirection, RandomWayPoint, GaussMarkov
        *** Association Control (AC) - mechanism that optimizes the use of the APs:
                llf (Least-Loaded-First)
                ssf (Strongest-Signal-First)"""
    net.startMobility(startTime=0, model='RandomWayPoint', max_x=120, max_y=120, min_v=0.3, max_v=0.5)
```

Figura 43: Código de la topología para probar la conexión y desconexión entre diferentes puntos de acceso

Como ya se ha modificado el código, simplemente habrá que guardarlo y ejecutarlo:

```
gedit ./wifiAssociationControl.py
```

En el momento en el que nuestro escenario este en ejecución, se va a realizar un ping entre las estaciones sta4 y sta3.

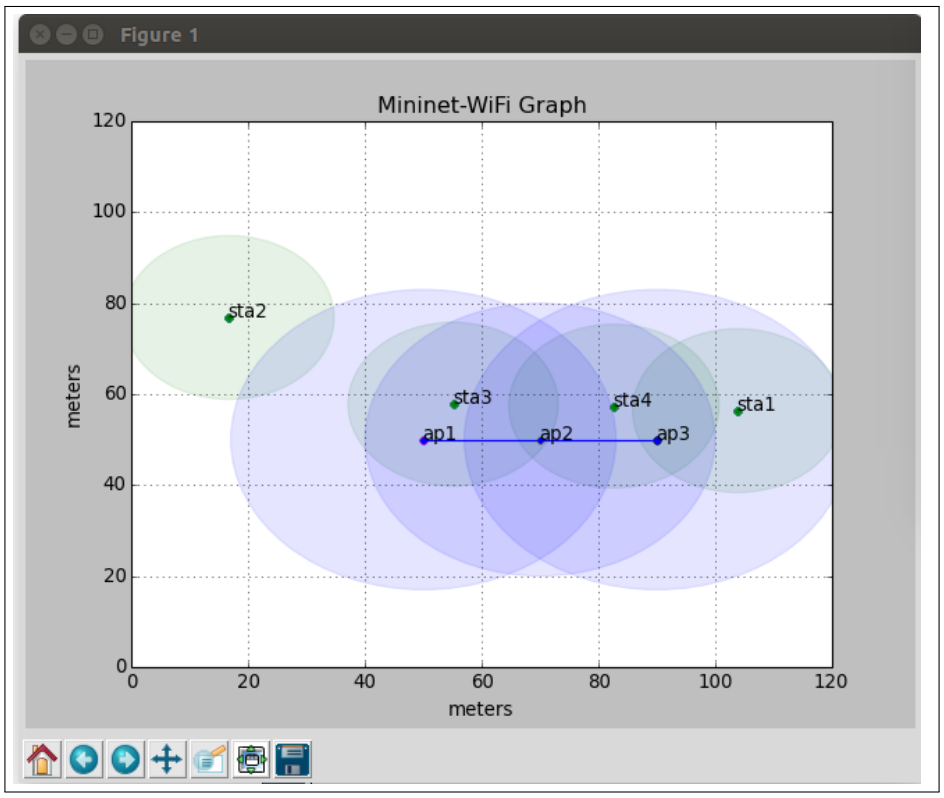


Figura 44: Visualización del momento del ping entre las estaciones sta4 y sta3

```
mininet-wifi> sta4 ping sta3
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=8.89 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=4.42 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=4.30 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=4.21 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=4.50 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=4.21 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=4.48 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=4.96 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=4.04 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=3.53 ms
64 bytes from 10.0.0.4: icmp_seq=11 ttl=64 time=3.55 ms
64 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=4.09 ms
64 bytes from 10.0.0.4: icmp_seq=13 ttl=64 time=3.36 ms
64 bytes from 10.0.0.4: icmp_seq=14 ttl=64 time=3.12 ms
64 bytes from 10.0.0.4: icmp_seq=15 ttl=64 time=2.86 ms
64 bytes from 10.0.0.4: icmp_seq=16 ttl=64 time=2.55 ms
```

Figura 45: Ping entre las estaciones sta4 y sta3

En las dos imagenes anteriores (Figura 44 y Figura 45) se observa que sta3 está conectada al punto de acceso ap1 mientras que sta4 está conectado a uno de los otros dos puntos de acceso, por lo que el ping que ésta estación envía alcanza su destino.

Por el contrario, el movimiento de las estaciones provoca que éstas cambien su posición y la sta3 pase a no estar en contacto con ningún punto de acceso por lo que para la sta4 será imposible conectarse a la sta3 y por esto el ping no alcanza su destino, como se puede observar en la figura 47.

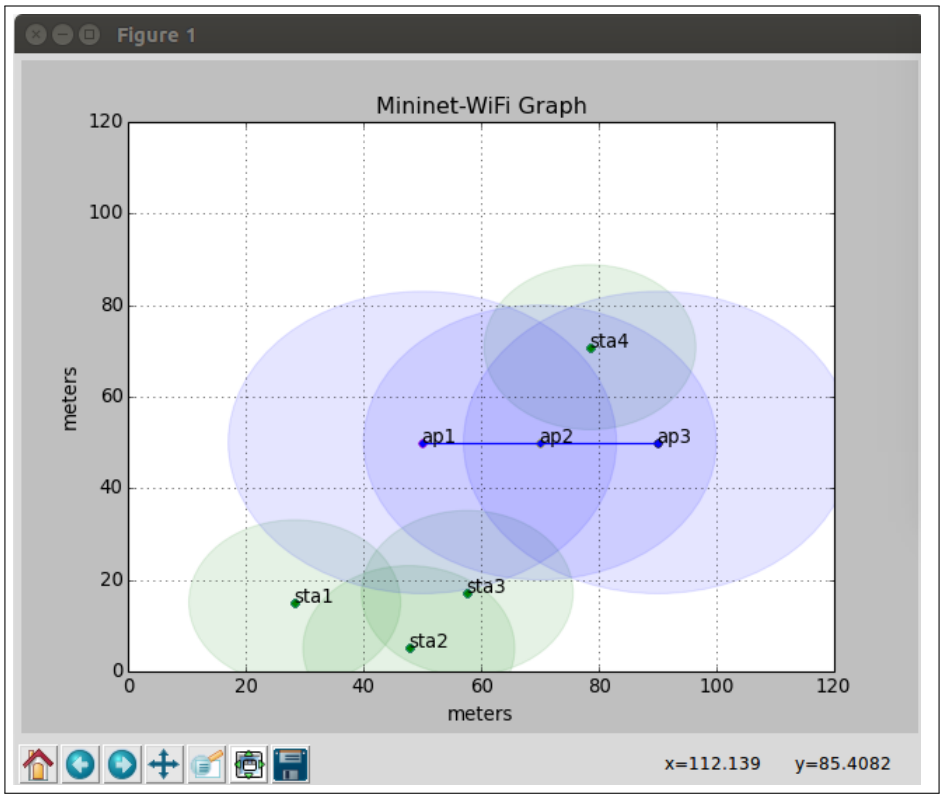


Figura 46: Visualización del momento del ping entre las estaciones sta4 y sta3 con ambas estaciones conectadas a puntos de acceso

```
64 bytes from 10.0.0.4: icmp_seq=33 ttl=64 time=4.85 ms
64 bytes from 10.0.0.4: icmp_seq=34 ttl=64 time=5.73 ms
From 10.0.0.4 icmp_seq=69 Destination Host Unreachable
From 10.0.0.4 icmp_seq=73 Destination Host Unreachable
From 10.0.0.4 icmp_seq=74 Destination Host Unreachable
From 10.0.0.4 icmp_seq=75 Destination Host Unreachable
From 10.0.0.4 icmp_seq=76 Destination Host Unreachable
From 10.0.0.4 icmp_seq=77 Destination Host Unreachable
From 10.0.0.4 icmp_seq=78 Destination Host Unreachable
From 10.0.0.4 icmp_seq=79 Destination Host Unreachable
From 10.0.0.4 icmp_seq=80 Destination Host Unreachable
```

Figura 47: Ping entre las estaciones sta4 y sta3 en el momento de que sta3 esté fuera del alcance de un punto de acceso

Por último, tras un tiempo la estación sta3 se reconecta a un nuevo punto de acceso, en este caso al punto de acceso ap3 y la estación sta4 al punto de acceso ap1, pero ésto no supone ningún problema y el ping alcanza su destino como se puede ver en la figura 49.

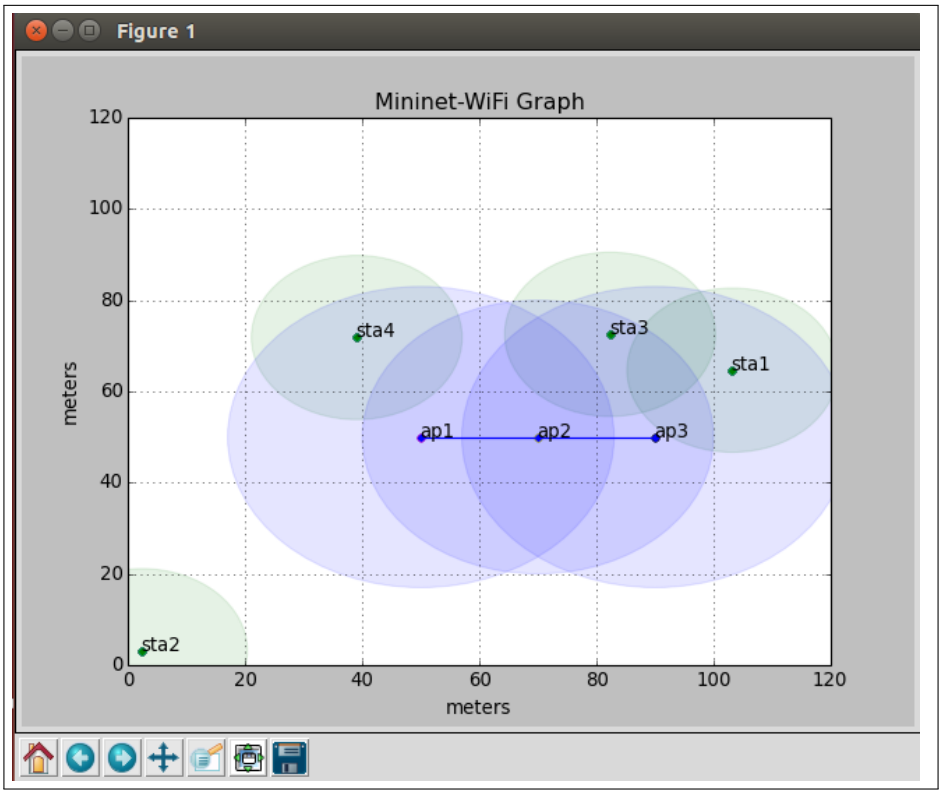
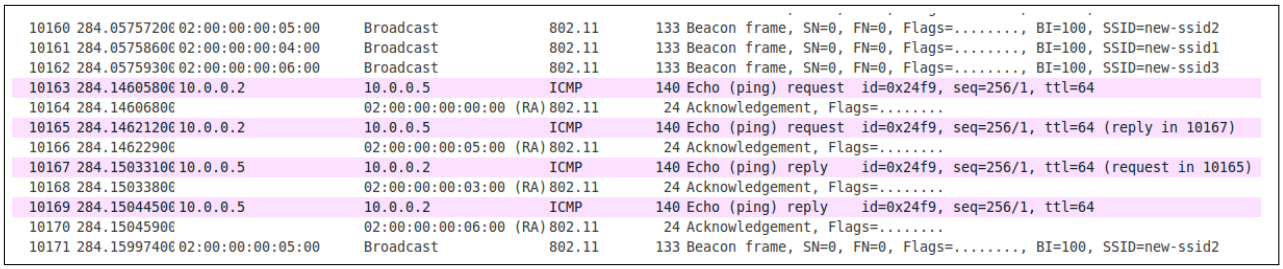


Figura 48: Visualización del momento del ping entre las estaciones sta4 y sta3 tras la reconexión a diferentes puntos de acceso del primer instante de conexión

```
64 bytes from 10.0.0.4: icmp_seq=32 ttl=64 time=2.29 ms
64 bytes from 10.0.0.4: icmp_seq=33 ttl=64 time=2.68 ms
64 bytes from 10.0.0.4: icmp_seq=34 ttl=64 time=3.15 ms
64 bytes from 10.0.0.4: icmp_seq=37 ttl=64 time=3.36 ms
64 bytes from 10.0.0.4: icmp_seq=38 ttl=64 time=3.33 ms
64 bytes from 10.0.0.4: icmp_seq=39 ttl=64 time=3.55 ms
64 bytes from 10.0.0.4: icmp_seq=42 ttl=64 time=4.92 ms
64 bytes from 10.0.0.4: icmp_seq=43 ttl=64 time=4.59 ms
64 bytes from 10.0.0.4: icmp_seq=44 ttl=64 time=5.27 ms
64 bytes from 10.0.0.4: icmp_seq=92 ttl=64 time=6.84 ms
64 bytes from 10.0.0.4: icmp_seq=93 ttl=64 time=6.97 ms
64 bytes from 10.0.0.4: icmp_seq=94 ttl=64 time=6.67 ms
64 bytes from 10.0.0.4: icmp_seq=95 ttl=64 time=7.28 ms
64 bytes from 10.0.0.4: icmp_seq=96 ttl=64 time=6.53 ms
64 bytes from 10.0.0.4: icmp_seq=97 ttl=64 time=6.46 ms
64 bytes from 10.0.0.4: icmp_seq=98 ttl=64 time=6.16 ms
64 bytes from 10.0.0.4: icmp_seq=99 ttl=64 time=6.04 ms
64 bytes from 10.0.0.4: icmp_seq=100 ttl=64 time=6.25 ms
64 bytes from 10.0.0.4: icmp_seq=103 ttl=64 time=4.84 ms
64 bytes from 10.0.0.4: icmp_seq=104 ttl=64 time=4.84 ms
64 bytes from 10.0.0.4: icmp_seq=105 ttl=64 time=5.28 ms
64 bytes from 10.0.0.4: icmp_seq=106 ttl=64 time=4.86 ms
64 bytes from 10.0.0.4: icmp_seq=107 ttl=64 time=5.24 ms
```

Figura 49: Ping entre las estaciones sta4 y sta3 en el momento de reconexión a los puntos de acceso

Como último análisis se van a realizar una serie de capturas sobre este último escenario en la interfaz hwsim0, para poder capturar en esta interfaz hay que seguir los pasos que se indicaron en el apartado anterior.



10160	284.0575720€	02:00:00:00:05:00	Broadcast	802.11	133	Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=new-ssid2
10161	284.0575860€	02:00:00:00:04:00	Broadcast	802.11	133	Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=new-ssid1
10162	284.0575930€	02:00:00:00:06:00	Broadcast	802.11	133	Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=new-ssid3
10163	284.1460580€	10.0.0.2	10.0.0.5	ICMP	140	Echo (ping) request id=0x24f9, seq=256/1, ttl=64
10164	284.1460680€		02:00:00:00:00:00 (RA)	802.11	24	Acknowledgement, Flags=........
10165	284.1462120€	10.0.0.2	10.0.0.5	ICMP	140	Echo (ping) request id=0x24f9, seq=256/1, ttl=64 (reply in 10167)
10166	284.1462290€		02:00:00:00:05:00 (RA)	802.11	24	Acknowledgement, Flags=........
10167	284.1503310€	10.0.0.5	10.0.0.2	ICMP	140	Echo (ping) reply id=0x24f9, seq=256/1, ttl=64 (request in 10165)
10168	284.1503380€		02:00:00:00:03:00 (RA)	802.11	24	Acknowledgement, Flags=........
10169	284.1504450€	10.0.0.5	10.0.0.2	ICMP	140	Echo (ping) reply id=0x24f9, seq=256/1, ttl=64
10170	284.1504590€		02:00:00:00:06:00 (RA)	802.11	24	Acknowledgement, Flags=........
10171	284.1599740€	02:00:00:00:05:00	Broadcast	802.11	133	Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=new-ssid2

Figura 50: Captura mediante Wireshark de tráfico WiFi

En la captura (Figura 50) se puede observar las tramas beacon de cada uno de los puntos de acceso, además del ping que se envió entre sta1 y sta4.

5

5. Interfaz física

En este apartado se realiza la conexión de una estación inalámbrica física, en este caso, un móvil, a través de una interfaz WiFi que funcionará como un punto de acceso, a una estación de un escenario de Mininet-WiFi.
La interfaz WiFi empleada es una TL-WN821N de TP-LINK.
Para convertir la interfaz WiFi en un punto de acceso se seguirán los siguientes pasos:

```
cd mininet-wifi
  cd hostapd
  cd hostapd
```

Una vez en el directorio mininet-wifi/hostapd/hostapd se debe entrar en la configuración:

```
gedit hostapd.conf
```

En este archivo se debe modificar:

```
interface=wlan1
   hw_mode=g
 driver=nl80211
   channel=7
ssid=Mininet-WiFi-AP
```

Una vez modificado el script de configuración, se probará si ya funciona la interfaz como punto de acceso. Para ello, se conecta la interfaz física, y se ejecuta el script hostapd.conf:

```
sudo hostapd hostapd.conf
```

En el caso de que no funcione es necesario utilizar los dos siguientes comandos:

```
sudo nmcli nm wifi off
sudo rfkill unblock wlan
```

Como se puede ver en la figura 51, ya está funcionando como punto de acceso.

Figura 51: Funcionamiento interfaz física como punto de acceso

Como se puede observar en la figura 52, el punto de acceso necesita una dirección IP.

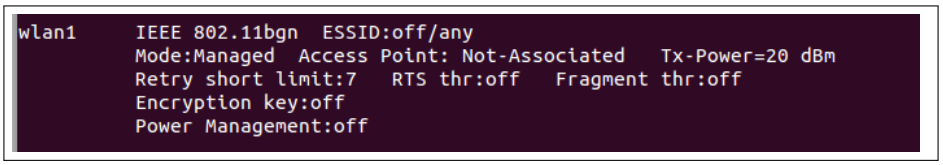


Figura 52: Características interfaz física

Se utilizará una asignación estática. Para ello se utiliza el siguiente comando:

```
ifconfig wlan1 192.168.0.1 netmask 255.255.255.0
```

También es necesario realizar la asignación de la dirección IP para el móvil como se puede ver en la figura 53.

Figura 53: Asignación estática direcciones IP

Una vez están las direcciones asignadas, se abrirá un escenario básico de Mininet-WiFi utilizando el comando:

```
sudo mn --wifi
```

Con este comando se abrirá el escenario más básico de Mininet-WiFi, con 1 punto de acceso y 2 estaciones.
En el caso de que al abrir el escenario el proceso que se estaba llevando acabo con hostapd se cerrase simplemente bastará con volver a ejecutar el script hostapd.conf.

En la ventana de cliente de Mininet-WiFi se va a crear un túnel para permitir que la interfaz wlan1 se comunique con el escenario de Mininet-WiFi. Para la creación del túnel hay que utilizar el siguiente comando:

```
sh ovs-vsctl add-port ap1 wlan1
```

Una vez creado el bridge ya es posible alcanzar desde el punto de acceso ap1 perteneciente al escenario de Mininet-WiFi el punto de acceso físico como se puede ver en la figura 54:

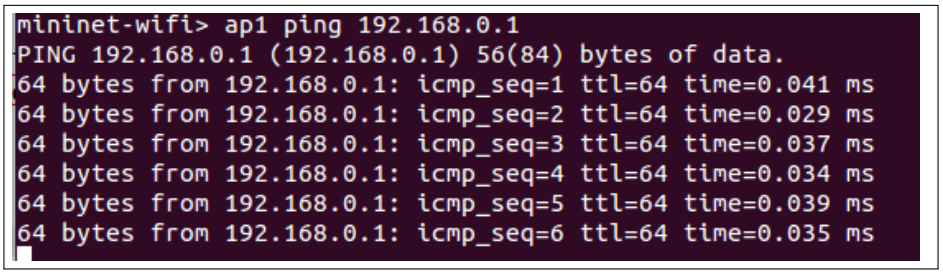


Figura 54: Ping entre puntos de acceso

6

6. Guión orientado a una práctica

Esta parte de la memoria está destinada a la realización de un guión para una práctica de laboratorio para alguna de las asignaturas impartidas por el Grupo de Ingeniería Telemática.

Está dividida en una introducción teórica y en un desarrollo práctico.

6.1. Introducción teórica

En esta parte se explicará todo lo necesario para posteriormente poder realizar el desarrollo práctico, se recomienda leer con atención este apartado, para evitar dudas posteriores durante la realización de la práctica en sí.

6.1.1. SDN y OpenFlow

Software-Defined Networking (SDN) es la separación en la gestión de los dispositivos de red del plano de control del plano de datos, es decir, separar el hardware del software, y donde el control de la red lo llevan a cabo dispositivos denominados controladores, en el caso de esta práctica OpenDayLight.
La arquitectura de SDN está formada por 3 capas que son accesibles desde interfaces de programación de aplicaciones (APIs):

- La Capa de Aplicación: consiste en las aplicaciones destinadas a los usuarios finales que serán los consumidores de los servicios de comunicaciones SDN. Permite a los servicios y aplicaciones simplificar y automatizar las tareas de configuración, provisión y gestionar nuevos servicios en la red, ofreciendo a los operadores nuevas vías de ingresos, diferenciación e inovación, además de suplir las necesidades de las diferentes aplicaciones a través de la programabilidad de la red SDN. El límite entre esta capa y la siguiente, la Capa de Control, es atravesado por la northbound API, en otras palabras, esta interfaz northbound sirve para conectar el controlador SDN a los servicios y aplicaciones por encima. Se trata de las interfaces más críticas ya que como se menciona anteriormente soportan a gran cantidad de aplicacones y servicios por encima de ellas.

- La Capa de Control: proporciona una funcionalidad centralizada de control que supervisa el comportamiento de la red de datos a través de una interfaz abierta; permite a los desarrolladores de aplicaciones utilizar capacidades de red, pero abstrayéndose de su topología o funciones. En relación a esta capa se debe hacer referencia al controlador SDN, ya que se trata de la entidad lógica de control encargada de traducir las peticiones de la aplicación SDN a las rutas de datos inferiores, dando a la capa de aplicación una visión abstracta de la red mediante estadísticas y posibles eventos. Se puede decir de los controladores que son el cerebro de este tipo de redes, ya que tienen control exclusivo sobre la forma de controlar y configurar los nodos de red para dirigir correctamente los flujos de tráfico; además de ésto, la arquitectura le permite generar un amplio rango de recursos

del plano de datos, lo cuál ofrece el potencial de unificar y simplificar su configuración.

- La Capa de Infraestructura: está constituida por los nodos de red que realizan la conmutación y encaminamiento de paquetes. Proporciona un acceso abierto programable a través de la southbound API, como por ejemplo OpenFlow. Las southbound API facilitan el control en la red, permitiendo al controlador realizar cambios dinámicos de acuerdo a las demandas en tiempo real y las necesidades.

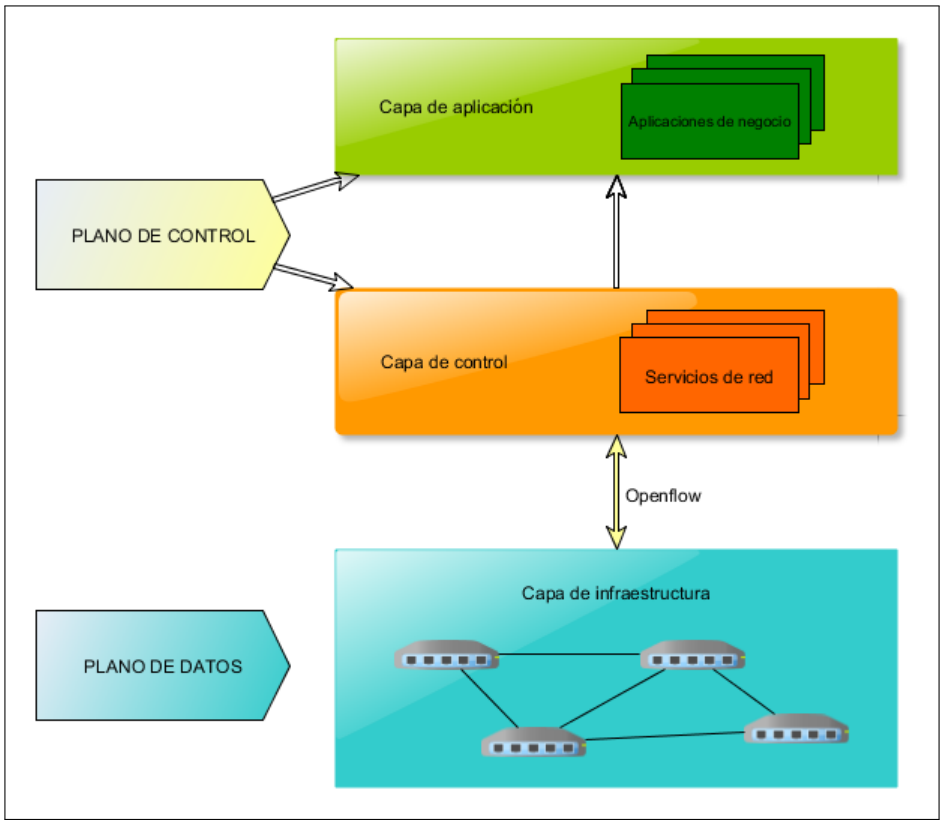


Figura 55: Arquitectura SDN

OpenFlow es el protocolo que permite a un servidor decirle a los conmutadores de red dónde enviar paquetes, permite al plano de control interactuar con el plano de datos. Con este protocolo se centralizan las decisiones de migración de paquetes, y ésto permite que la red se pueda programar independientemente de los conmutadores individuales y el equipo de centro de datos.

Un switch OpenFlow está formado al menos por tres partes, una o más Flow Tables, una Group Table, y un OpenFlow Channel para la conexión con el controlador.

- Tabla de flujos: con una acción asociada a cada entrada de la tabla, indicando al switch cómo debe procesar ese flujo.

- Tabla de grupos: consiste en un grupo de entradas. Los grupos proveen una manera eficiente de indicar que el mismo conjunto de acciones deben ser llevadas a cabo por múltiples flujos. Por esta razón, es útil para implementar tanto multicast y unicast.

- OpenFlow Channel: es la interfaz que conecta cada switch OpenFlow con un controlador. A través de esta interfaz el controlador configura y maneja el switch, recibe eventos desde el switch, y envía paquetes fuera del switch. Entre el datapath y el OpenFlow Channel, la interfaz es implementación específica, sin embargo, todos los mensajes del OpenFlow Channel deben estar formateados de acuardo al protocolo OpenFlow. Normalmente está encriptado usando TLS, pero puede estar directamente sobre TCP.

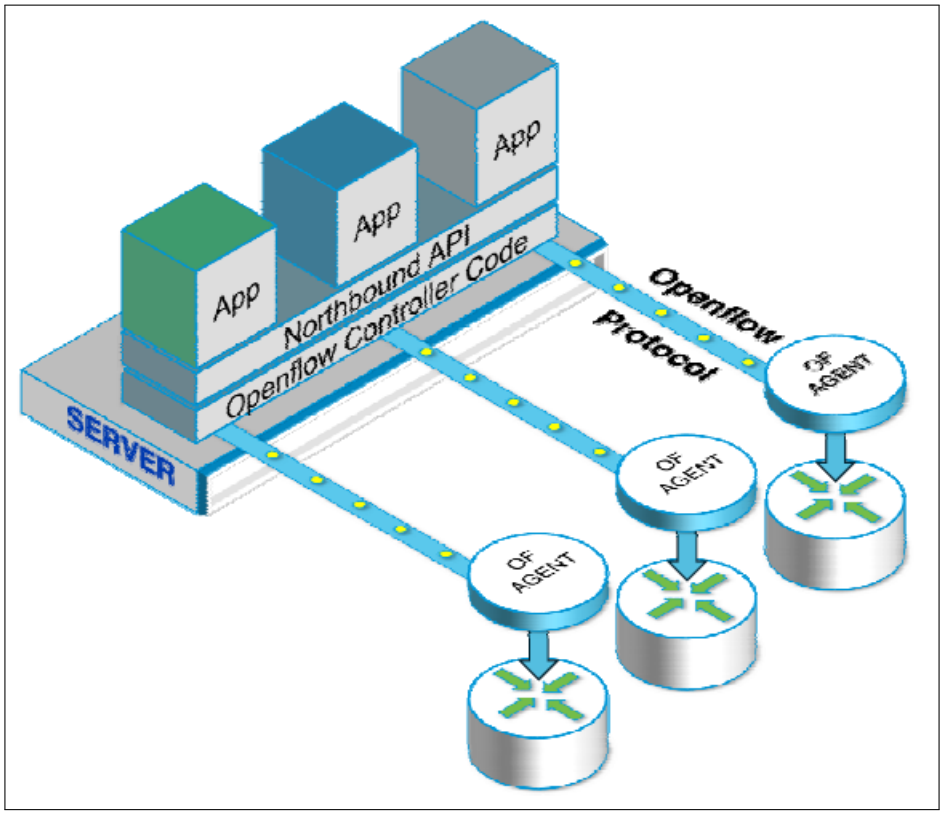


Figura 56: Arquitectura OpenFlow

Por lo tanto, cabe destacar que OpenFlow no es SDN, es simplemente un elemento que forma parte de la arquitectura SDN.

6.1.2. Controlador OpenDayLight

Se trata de un controlador OpenFlow. Toda comunicación entre las aplicaciones y cualquier dispositivo tiene que pasar a través de él. El protocolo OpenFlow conecta el software del controlador con los dispositivos de red de forma que éste pueda decir a los switches dónde enviar los paquetes. El controlador usa el protocolo OpenFlow para configurar los dispositivos de red y elegir el mejor camino para el tráfico.
Centrandonos en el controlador ODL, éste nos ofrece acciones como:

- Control centralizado de los dispositivos físicos y virtuales en la red.
- Control de los dispositivos con estándares y protocolos abiertos.
- Proporciona abstracción de alto nivel de sus capacidades para que los ingenieros de redes y los desarrolladores puedan crear nuevas aplicaciones para personalizar la configuración y administración de redes.

Los casos de uso para SDN son los siguientes:

- Centralizado de monitorización de red, gestión y coordinación.
- Gestión proactiva de redes e ingeniería de tráfico.
- NUBE- gestionar tanto la superposición virtual y la capa base física debajo de ella.
- Encadenamiento de paquetes a través de las diferentes máquinas virtuales.

OpenDayLight posee una interfaz gráfica con diferentes apartados:

- Topology
- Nodes
- Yang UI
- Yang Visualizer

Con el primero de ellos, Topology, se puede observar la topología de red que se emula con Mininet. Con el siguiente, Nodes, se puede ver los nodos que forman la red, es decir, la información de cada switch. A continuación, se encuentra Yang UI, se trata de una estructura de modelado de datos, que provee funcionalidades en los switches de SDN. Se puede utilizar para obtener información del almacén de datos o para construir comandos y construir la información de éste almacén, cambiando así la configuración de la red. Dentro de Yang UI se encuentra un apartado llamado API, pinchando en él, y dentro de este en Inventory API, aparece un inventario de información de nuestra red, como son las estadisticas, los puertos y los nodos entre otros.

6.1.3. Mininet-WiFi

Se trata de una herramienta del emulador de SDN de Mininet. Los desarrolladores de Mininet-WiFi aumentaron las funcionalidades de Mininet añadiendo estaciones y puntos de acceso inalámbricos basados en un controlador 80211_hwsim. Además, añaden clases para soportar estos dispositivos inalámbricos en los escenarios de Mininet, también se añaden atributos como la posición y el movimiento relativo de las estaciones móviles y los puntos de acceso. Mininet-WiFi extiende el código base de Mininet, añadiendo o modificando clases y scripts. Por lo tanto, Mininet-WiFi añade nuevas funcionalidades y conserva aquellas que tenía Mininet original.

6.1.4. Como utilizar Mininet-WiFi

En Mininet-WiFi aparecen diferentes comandos para poder analizar tanto la topología como el tráfico de la red, entre otros se encuentran los siguientes:
Para enviar un ping entre estaciones se utiliza el comando siguiente:

```
sta1 ping sta2
```

Si lo que se quiere es enviar un ping entre todos los nodos de la red se utilizará:

```
pingall
```

Si se quiere saber cuáles son los puntos de acceso visibles para cada estación se usará el comando siguiente:

```
sta1 iw dev sta1-wlan0 scan | grep ssid
```

Si lo que se quiere saber es con qué punto de acceso esta conectada la estación se utilizará el comando:

```
sta1 iw dev sta1-wlan0 link
```

Para obtener datos de las estaciones o puntos de acceso se disponen de diferentes comandos dependiendo de los datos que se quieran obtener:
- La posición:

```
position ap1
```

- La distancia:

```
distance ap1 sta2
```

- Información(número de asociaciones, potencia,SSID)

```
info ap1
```

En relación a los scripts de Python dependiendo lo que se quiera crear se utilizaran unas líneas de código u otras: - Para crear una estación:

```
net.addStation('sta1')
```

Con esto se añade una estación con todos los parámetros por defecto.
- Para crear un punto de acceso con todos los valores por defecto y con SSID:

```
net.addBaseStation('ap1', ssid='new_ssid')
```

- Para crear un enlace:

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

- Para crear estaciones con diferentes características:

```
net.addStation( 'sta1', passwd='123456789a', encrypt='wpa2',
mac='00:00:00:00:00:02', ip='10.0.0.2/8', position='50,30,0' )
```

- En los enlaces también se pueden añadir características:

```
net.addLink( ap1, sta1, bw='11Mbps', loss='0.1%', delay='15ms')
```

- Para realizar las asociaciones de control en una red estática se puede usar el método associationControl (ssf(Strongest-Signal-First) y llf (Least-Loaded-First)):

```
net.associationControl('ssf')
```

- Para añadir un modelo de movilidad de las estaciones hay disponibles los siguientes: `RandomWalk`, `TruncatedLevyWalk`, `RandomDirection`, `RandomWayPoint`, `GaussMarkov`, `ReferencePoint`,`TimeVariantCommunity`.
Un ejemplo sería:

```
net.startMobility(startTime=0, model='RandomDirection', max_x=60,
            max_y=60, min_v=0.1, max_v=0.2, AC='llf')
```

- Para añadir la propagación hay disponibles los siguientes: `friisPropagationLossModel` , `twoRayGroundPropagationLossModel`, `logDistancePropagationLossModel`.
Un ejemplo sería:

```
net.propagationModel('friisPropagationLossModel', sL=2)
```

6.2. Desarrollo práctico

Para el desarrollo de la práctica se empleará una máquina virtual, el nombre de ésta es Mininet-Wifi y la contraseña para entrar en ella es 123456.
Una vez dentro de la maquina virtual es necesario que se entre en Mininet-WiFi para poder llevar a cabo la práctica. Es necesario seguir los siguientes pasos:

```
cd mininet-wifi
  cd examples
```

Una vez aquí ya se podrán ejecutar los escenarios necesarios para la realización de la práctica. Pero previamente, es necesario iniciar el controlador remoto OpenDayLight, por lo que se abre otra ventana de la terminal y se siguen los pasos siguientes:

```
cd distribution-karaf-0.4.0-Beryllium
./bin/karaf
```

Ahora ya se está preparado para empezar el estudio y análisis de los diferentes escenarios.
Se empezará ejcutando el escenario 2AccessPoints.py:

```
./2AccessPoints.py
```

En el script de este escenario se puede observar que este escenario está formado por dos puntos de acceso y cuatro estaciones, además de un controlador.
Se empezará por ejecutar una serie de comandos:
El primer comado será:

```
info stax
```

El segundo comando ejecutado será:

```
stax iw dev sta1-wlan0 scan | grep ssid
```

(Sustituir la x por el número de estación del que se quiera saber la información)

Responder a las siguientes preguntas:

- Indicar a que punto de acceso está conectado cada estación.
- Indicar que puntos de acceso son visibles desde cada estación.
- DIbujar el esquema de la red.

Una vez analizado este escenario se procederá a analizar otro segundo escenario en el que se utiliza la característica más importante de Mininet-WiFi, la movilidad.
Se empezará por analizar el script de este escenario. Para ello, es necesario abrir el script:

```
gedit wifiMobility.py
```

Tras ésto se pasará a ejecutarlo y contestar una serie de preguntas.
Responda a las siguientes preguntas:

- Explicar las características principales del script y el tipo de movimiento empleado.
- Realizar un ping entre las estaciones y explicar que ocurre al principio y al final del movimiento.

El siguiente escenario a analizar será wifiMobilityModel.py en él se observará el funcionamiento de uno de los modelos de movilidad además de analizar el tráfico en el punto de acceso.
Para analizar el tráfico se usará el siguiente comando:

```
dpctl dump flows
```

Responder a las siguientes preguntas:

- Observar el funcionamiento del modelo RandomDirection.
- Analizar el tráfico entre sta1 y sta2 (indicando el porcentaje de paquetes partidos en 1 minuto, la dirección MAC del destino y del origen)

Ahora el escenario que se va a analizar será wifiAsocciationControl.py, en este escenario se usará otro modelo de movilidad, en este caso será RandomWayPoint, y también se usará por primera vez en la práctica los modelos de asociación, utilizando ssf (Strongest-Signal-First).

- Analizar el script y explicar las principales caracteristicas de este.
- Observar el funcionamiento del modelo de movilidad RandomWayPoint.
- Indicar a que punto de acceso está conectado cada estación en el momento inicial y observar como cambia con el tiempo.
- Indicar que puntos de acceso son visibles desde cada estación y observar como cambia con el tiempo.
- Observar el tráfico entre las sta1 y sta2, observando los momentos de desconexión y conexión con los diferentes puntos de acceso.
- Analizar el tráfico mediante el comando dpctl dump flows.

El último escenaro será wifiPropagationModel.py en este escenario se probarán diferentes métodos de propagación. Lo primero será abrir el script con el comando gedit. Una vez se haya visto el modelo de propagación que se está utilizando se ejecutará. Tras haber ejecutado este modelo se mirará el nivel de señal mediante el comando info.
Responder a las siguientes cuestiones:

- Cuál de los modelos de propagación muestra mayor señal en la sta1.
- Cuál de los modelos de propagación muestra menor señal en la sta2.

Como último ejercicio se pide realizar un escenario con las siguientes características:

- 3 estaciones moviles (sta1, sta2 y sta3)
- 1 estación fija (h1)
- 2 puntos de acceso(ap1 y ap2)
- 1 controlador remoto
- Modelo de movilidad: RandomDirection
- Modelo de asociación: ssf

Cuando se disponga del script creado se va a realizar el análisis del tráfico, pero en este caso se realizará mediante Wireshark. La captura se realizará en tres interfaces diferentes, la primera interfaz en la que se capturará será en una interfaz de las de uno de los puntos de acceso para observar ahí el tráfico, tras ésto se analizará el tráfico en la interfaz de Loopback, y por último se analizará en la interfaz hwsim0.
Para poder capturar en la interfaz hwsim0 es necesario activarla previamente, para ello hay que escribir en la ventana de cliente lo siguiente:

```
sh ifconfig hwsim0 up
```

Se pide:

- Explicar las diferencias de capturar en una interfaz u otra.
- Analizar los paquetes que se encuentran en las diferentes capturas.

7

7. Conclusiones y líneas futuras

7.1. Conclusiones

Tras el estudio de SDN y OpenFlow para posteriormente llevarlo a la práctica utilizando la herramienta Mininet-WiFi se puede decir que se han alcanzado los objetivos que fueron planteados previamente a la realización del trabajo. A pesar de diferentes problemas y dificultades que han ido surgiendo a lo largo de la realización del mismo, no ha sido impedimento para poder observar el funcionamiento del protocolo OpenFlow, como así, para la utilización del controlador OpenDayLight y las ventajas que ha aportado. También se ha utilizado MiniEdit ya que parecía la forma más fácil de empezar a entender y manejar la herramienta para posteriormente, pasar al uso de Mininet-WiFi, donde se ha llevado a cabo tanto la realización de escenarios con puntos de acceso que emplean el protocolo OpenFlow como capturas de tráfico mediante Wireshark para poder observar más detalladamente cómo están constituidas las tramas.
Una vez entendido SDN y OpenFlow, así como, Mininet-WiFi, se realizó un guión en el que se pretende que el alumno lleve a cabo diferentes simulaciones de los escenarios para afianzar los conceptos sobre los que ha tratado el proyecto. Esta práctica no presenta gran dificultad ya que es muy guiada.
Haciendo referencia a alguna de las dificultades que se han encontrado en la realización del proyecto y por lo que no se ha podido profundizar más son, por ejemplo, lo reciente de Mininet-WiFi, ya que había poca información al iniciar el proyecto, también surgieron problemas relacionados con la partición del disco duro y la desconfiguración de los elementos instalados, a estos problemas, hay que sumarle problemas en las simulaciones, problemas desconocidos, ya que sobre el mismo escenario, sin ninguna modificación, con reiniciar ya funcionaban.

7.2. Líneas futuras

A partir de este proyecto pueden surgir nuevas ideas para continuar con el análisis de OpenFlow y SDN. Una de las ideas podría ser la utilización de varias interfaces físicas WiFi conectadas entre si, junto a escenarios de Mininet-WiFi, todo ello conectado entre sí y pudiendo comunicarse entre ellos. También se podría profundizar en el estudio de los modelos tanto de movilidad como de propagación. Se podría utilizar el controlador remoto OpenDayLight para el control de estaciones físicas. Ésto son solo algunas ideas, ya que pueden surgir gran cantidad de ideas.

Referencias

[1] Software Defined Networking definition. http://www.opennetworking.org/sdn-resources/sdn-definition, 20 de Junio de 2016.

[2] Introducción a las Redes Definidas por Software. searchdatacenter.techtarget.com/es/definición/redes-definidas-por-software-sdn, 20 de Junio de 2016.

[3] SDN. https://www.wikipedia.org/wiki/redes-definidas-por-software, 20 de Junio de 2016.

[4] Redes Definidas por Software. www.ramonmillan.com/tutoriales/sdnredesinteligentes.php, 20 de Junio de 2016.

[5] Introducción a OpenFlow. www.ramonmillan.com/documentos/bibliografía/openflowenabled, 20 de Junio de 2016.

[6] What's Software Defined Networking? http://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/, 20 de Junio de 2016.

[7] Open Networking Foundation. Openflow switch specification version 1.3.1, 6 de Septiembre de 2012.

[8] OpenFlow. https://es.wikipedia.org/wiki/openflow, 23 de Junio de 2016.

[9] OpenFlow controller. http://searchsdn.techtarget.com/definition/openflow-controller, 28 de Junio de 2016.

[10] Controladores SDN: elementos para su selección y evaluación. http://revistatelematica.cujae.edu.cu/index.php/tele/article/viewfile/164/153, 28 de Junio de 2016.

[11] OpenDayLight el futuro de las redes definidas por software. blog.desdelinux.net/opendaylight-el-futuro-de-las-redes-definidas-por-software-sdn/, 26 de Junio de 2016.

[12] OpenDayLight. Opendaylight user guide, 26 de Junio de 2016.

[13] NOX y POX. http://www.noxrepo.org/, 30 de Junio de 2016.

[14] Introduction to Mininet. https://github.com/mininet/mininet/wiki/introduction-to-mininet#what, 26 de Junio de 2016.

[15] Mininet-Wifi. www.brianlinkletter.com/mininet-wifi-software-defined-network-emulator-supports-wifi-networks/, 26 de Junio de 2016.

[16] Introduction to MiniEdit. http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/, 20 de Julio de 2016.

[17] Using the OpenDaylight SDN Controller with the Mininet Network Emulator. http://www.brianlinkletter.com/using-the-opendaylight-sdn-controller-with-the-mininet-network-emulator/, 26 de Junio de 2016.

[18] Uso de Mininet. http://www.brianlinkletter.com/mininet-test-drive/, 21 de Julio de 2016.

[19] IEEE Fernando M.V.Ramos Member IEEE Paulo Verissimo Fellow IEEE Christian Esteve Rothenberg Member IEEE Siamak Azodolmolky Senior Member IEEE Diego Kreutz, Member and IEEE Steve Uhlig, Member. Software-defined networking:a comprehensive survey, 8 de Octubre de 2014.

[20] SCC. Sdn redes definidas por software: http://es.scc.com/news/sdn-redes-definidas-por-software/, 8 de septiembrede 2016.

[21] Celia Villarrubia Datacenter Dynamics. Sdn, la evolución que necesitaba la red: http://www.datacenterdynamics.es/focus/archive/2013/03/sdn-la-evolución-que-necesitaba-la-red, 26 de Julio de 2016.

[22] Cinco protocolos sdn que no son openflow: http://searchdatacenter.techtarget.com/es/cronica/cinco-protocolos-sdn-que-no-son-openflow, 8 de Septiembre de 2016.

[23] Ramon Fontes. Modelos de propagación: Mininet-wifi https://github.com/intrig-unicamp/mininet-wifi/blob/master/examples/wifipropagationmodel.py, 21 de Julio de 2016.

[24] André Panisson. Modelos de movilidad: Mininet-wifi https://github.com/intrig-unicamp/mininet-wifi/blob/master/mininet/mobility.py, 21 de Julio de 2016.

Acrónimos

- **AAA** (Authentication, Authorization and Accounting)
- **AC** (Asociation Control)
- **API** (Application Programming Interface)
- **ARP** (Address Resolution Protocol)
- **BGP** (Border Gateway Protocol)
- **ETH** (Ethernet)
- **GUI** (Graphical User Interface)
- **ICMP** (Internet Control Message Protocol)
- **IEEE** (Institute of Electrical and Electronics Engineers)
- **IP** (Internet Protocol)
- **ITU** (International Telecommunication Union ITU)
- **JSON** (JavaScript Object Notation)
- **LIS** (Location Information Server)
- **LLDP** (Link Layer Discovery Protocol)
- **MAC** (Media Access Control)
- **MPLS-TP** (Multiprotocol Label Switching - Transport Profile)
- **NETCONF** (Network Configuration Protocol)
- **ODL** (OpenDayLight)
- **OF** (OpenFlow)
- **OFS** (OpenFlow Switch)
- **OSPF** (Open Shortest Path First)
- **OvS** (Open vSwitch)
- **OVSDB** (Open vSwitch Database)
- **PCEP** (Path Computation Element Communication Protocol)
- **REST** (Representational State Transfer)
- **SDN** (Software-Defined Networking)
- **SNMP** (Simple Network Management Protocol)
- **SSID** (Service Set Identifier)
- **TCP** (Transmission Control Protocol)
- **TLS** (Transport Layer Security)
- **UI** (User Interface)

- **URL** (Uniform Resource Locator)
- **XML** (eXtensible Markup Language)
- **XMPP** (Extensible Messaging and Presence Protocol)