Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

“Aplicaciones de SDN/NFV en Redes Inalámbricas de Área Local”

Autor: Ramón Zadiel Estrada de la Torre.

Tutor: MSc.Arelys Ramos Fleytes.

Santa Clara

2017

"Año 59 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

“Aplicaciones de SDN/NFV en Redes Inalámbricas de Área Local”

Autor: Ramón Zadiel Estrada de la Torre.

E-mail: redelatorre@uclv.cu

Tutor: MSc.Arelys Ramos Fleytes.

E-mail: arelys@uclv.edu.cu

Departamento de Telecomunicaciones y Electrónica.

Facultad de Ingeniería Eléctrica.

Santa Clara

2017

"Año 59 de la Revolución "

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

Borraré de mi vocabulario palabras como "abandonar la idea"," no puedo"," irrealizable", "sin esperanza", etc.…Porque son palabras de personas que no tienen fe en sí mismas, ni en Dios.

Og Mandino

DEDICATORIA

A mis progenitores y al Raymond ’s team.

AGRADECIMIENTOS

A mis padres, a mi tutora y a todo el que me apoyó de una forma u otra para realizar este trabajo y para llegar a obtener el título de ingeniero.

TAREA TÉCNICA

- ✓ Análisis de los fundamentos teóricos de las Redes Definidas por Software (SDN) y de la Virtualización de las Funciones de Red (NFV).
- ✓ Estudio de los controladores disponibles en la industria para redes inalámbrica.
- ✓ Análisis de las herramientas de simulación Mininet y Mininet -WiFi.
- ✓ Identificación de las principales aplicaciones SDN/NFV en las redes inalámbricas.
- ✓ Desarrollo de una aplicación de red inalámbrica con SDN/NFV con diferentes escenarios de simulación.

Firma del Autor

Firma del Tutor

RESUMEN

Debido a la enmarcación que están teniendo actualmente las redes de telecomunicaciones y las limitantes que surgen producto de la gran demanda que existe y del crecimiento exponencial del tráfico que circula por la red, ha venido surgiendo en los últimos años una alternativa a las redes tradicionales, las redes definidas por software (SDN, por sus siglas en inglés), una arquitectura de red emergente que se destaca como capaz de ofrecer un control más flexible y dinámico a la hora de gestionar el diseño y funciones de red, concepto que está revolucionando las Telecomunicaciones . A la par con las SDN, pero más reciente aún, se viene investigando otro concepto novel, las redes inalámbricas de área local definidas por software (SDWLAN, por sus siglas en inglés). Estas tecnologías ofrecen un amplio panorama de investigación y se prevé que en un futuro sean la base de una nueva era de las comunicaciones. Tomando todo esto como motivación en el presente trabajo se realizará un estudio de las SDN y específicamente de las SDWLAN.

Primeramente, se analizan los fundamentos teóricos de este tipo de redes, incluyendo arquitectura, protocolo OpenFlow, principales controladores, entre otros. Se fundamentan dos de las principales aplicaciones de las SDWLAN, posteriormente se caracterizan las herramientas de simulación más utilizadas y se analizan las principales funcionalidades de Mininet-WiFi, la herramienta de simulación por excelencia de las SDWLAN. Finalmente se crean diferentes escenarios de SDWLAN mediante scripts de Python, utilizando el controlador POX, se simulan en Mininet-WiFi y se comprueba su correcto funcionamiento en cada caso.

GLOSARIO DE ACRÓNIMOS

AC: Association Control, Control de Asociación
ACK: Acknowledgment
ACL: Access Control List, Listas de Control de Acceso
AP : Access Point, Punto de Acceso
API :Application Programming Interface, Interfaz de Programación de Aplicación
ARP :Address Resolution Protocol, Protocolo de Resolución de Direcciones
ETH :Ethernet
GUI :Graphical User Interface, Interfaz Gráfica de Usuario
ICMP :Internet Control Message Protocol, Protocolo de Mensajes de Control de Internet
IEEE :Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos
IP :Internet Protocol ,Protocolo de Internet
ITU :International Telecommunication Union ITU, Unión Internacional de Telecomunicaciones.
JSON :JavaScript Object Notation , Notación de Objeto de JavaScript
LIS :Location Information Server, Servidor de Información de Ubicación
LLDP :Link Layer Discovery Protocol,Protocolo de Descubrimiento de Capa de Enlace
LVAP :Light Virtual Access Point, Punto de Acceso Virtual Ligero
MAC :Media Access Control, Control de Acceso al Medio
NETCONF :Network Configuration Protocol , Protocolo de Configuración de Red
NIC :Network Interface Card, Tarjeta de Interfaz de Red.
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)
QoS :Quality of Service, Calidad de Servicio
REST (Representational State Transfer)
SDN :Software-Defined Networking ,Redes Definidas por Software
SDWLAN :Software Define Wireless Local Area , Redes de Area Local Definidas por Software
SNMP :Simple Network Management Protocol, Protocolo de Administración Simple de Red
SSID :Service Set Identifier, Identificador de Servicios de Conjunto
STA:Station, Estación
TCP :Transmission Control Protocol, Protocolo de Control de Transmisión
TLS :Transport Layer Security, Seguridad de Capa de Transporte
UI :User Interface, Interfaz de Usuario
URL :Uniform Resource Locator,Localizador de Recursos Uniformes

ÍNDICE

INTRODUCCIÓN

Durante muchos años se viene generando un crecimiento fuerte y sostenido de las redes de telecomunicaciones, resultado de una demanda cada vez mayor por parte de los usuarios. Este crecimiento, desordenado por basarse en principios y modelos no escalables, demostró que las infraestructuras y arquitecturas tradicionales presentan serias limitaciones de adaptación, flexibilidad y crecimiento: por lo que se ha hecho necesario realizar un cambio de enfoque sobre las arquitecturas de redes.

El problema radica en el esquema de control tradicional, el cual se basa en un enfoque no centralizado. La administración de los diferentes equipos por separado, teniendo únicamente un plano de control distribuido, no provee la flexibilidad, escalabilidad, y dinámica que las redes actuales requieren. Estas topologías tradicionales han llegado a ser extremadamente complejas y difíciles de abarcar. Se plantea un nuevo paradigma, basado en principios abstractos y fundamentales. El estudio y análisis de los métodos de aplicación de los diversos protocolos se realiza una única vez, creando abstracciones y utilizándolas para explicar y modificar el funcionamiento de una red. De esta forma, la red se pone al servicio de los operadores y programadores, y las diferentes funcionalidades se programan, configuran y operan a un nivel de control centralizado, sin tener que lidiar con la distribución de la inteligencia. Se proponen nuevas arquitecturas de red que sean flexibles, escalables, programables y capaces de soportar múltiples servicios; que logren optimizar la forma en la que se opera, se diseña e interactúa con las redes. Junto a ello, se busca acompañar el creciente despliegue de servicios de tipo “cloud computing” y la virtualización de las funciones de red. Es entonces SDN (Software Defined Networking) un enfoque de arquitectura de red que brinda la habilidad de programar directamente operaciones de red, usando lenguajes y sistemas operativos estándar. SDN permite realizar una transición, desde modelos de redes

tradicionales hacia redes programables, que brindan flexibilidad, agilidad, optimización y virtualización.

La funcionalidad SDN les permite a los operadores de red utilizar conceptos de programación para implementar técnicas de administración y orquestación automáticas, aplicar diversas configuraciones en múltiples equipos, y desacoplar las aplicaciones que realizan estas operaciones de los sistemas operativos de los dispositivos de red. SDN permite a las aplicaciones interactuar dinámicamente con los diferentes procesos de la red, usando estadísticas y diferentes parámetros recibidos de la misma, junto con requerimientos específicos, para desplegar de forma dinámica nuevos esquemas y configuraciones.

A la par con SDN se viene desarrollando el estudio de otro concepto novel y del cual se conoce muy poco aún, SDWLAN (Software Defined Wireless Local Area Networks) Redes Inalámbricas de Área Local Definidas por Software, tecnología de red emergente en la cual se centrará el presente proyecto de investigación, las SDWLAN permiten gestionar de forma flexible los dispositivos inalámbricos empleando un controlador ,permiten crear reglas en el flujo de datos y controlar el comportamiento de los parámetros de los dispositivos inalámbricos(canales, control de acceso, potencia ,etc.).Está demostrado que existe una mejora significativa en el rendimiento de una red al utilizar un controlador centralizado. Un controlador SDN puede obtener información de los dispositivos alrededor de cada punto de acceso que contenga en su firmware el protocolo OpenFlow, permitiendo conocer toda la topología de la red. Con esa información, es posible reducir la interferencia de los puntos de acceso adyacentes mediante el control de la potencia. Por otra parte, SDWLAN puede hacer que múltiples puntos de acceso actúen como un solo gran punto de acceso inalámbrico virtual, para evitar desconexiones de enlaces innecesarios.

Motivado por la importancia que está teniendo hoy en día las SDWLAN se plantea como problema científico de esta investigación el siguiente:

¿Cuáles son los principales retos que enfrentan las SDWLAN en la actualidad?

Como objetivo general se presenta el siguiente:

Desarrollar aplicaciones de NFV/SDN en las redes inalámbricas de área local.

Con vistas a alcanzar este objetivo general se deben completar una serie de objetivos específicos que se resumen en los siguientes puntos:

- Analizar las aplicaciones NFV/SDN en las redes inalámbricas.

- Analizar la factibilidad de implementar SDN en redes inalámbricas.
- Simular en una herramienta de software diferentes escenarios de una red SDWLAN.

Este proyecto de investigación está estructurado en tres capítulos. El capítulo 1 aborda todo lo relacionado con las SDN, el protocolo OpenFlow, ventajas, arquitectura, usos y se hace una breve introducción a las SDWLAN. En el capítulo 2 se fundamentan dos de las aplicaciones más útiles y novedosas de SDN en redes WLAN, se realiza una caracterización de las principales herramientas de simulación empleadas y se presenta un breve tutorial con las principales funcionalidades de Mininet-WiFi, la herramienta de simulación por excelencia de las SDWLAN. Por último, en el capítulo tres se realizan las simulaciones llevadas a cabo en diferentes escenarios de SDWLAN y se analizan los resultados obtenidos.

CAPÍTULO 1. INTRODUCCIÓN A SDWLAN

En este capítulo se recogen los aspectos fundamentales de las Redes Definidas por Software(SDN), partiendo de la necesidad de un nuevo paradigma de red, los beneficios que nos brinda la misma, sus aplicaciones principales, la arquitectura y los controladores más usados y al final del capítulo se hace una breve introducción a las Redes Inalámbricas Definidas por Software(SDWN).

1.1. Importancia de un nuevo paradigma de red.

Las arquitecturas de red existentes no satisfacen al máximo las necesidades de los usuarios, empresas y operadores de hoy en día, los diseñadores de redes se encuentran restringidos por las limitaciones de las redes actuales, entre las que se encuentran:

- **Escalabilidad Limitada:** La red se vuelve más complicada con la adición de nuevos dispositivos que se deben configurar y gestionar manualmente.
- **Dependencia de proveedores:** Los operadores tratan de desplegar nuevos servicios de respuesta rápida a las necesidades cambiantes del negocio. Sin embargo, su capacidad de respuesta se ve obstaculizada por los ciclos de los vendedores de productos. La falta de interfaces estándar y libres limita la capacidad de los operadores para adaptar la red a sus entornos individuales. En redes pequeñas, la escalabilidad no es bien estimada ya que no surgen las necesidades adecuadas para esto, pero en la redes modernas, donde la flexibilidad es algo de todos los días, y con el advenimiento de los usuarios móviles, se crea la necesidad de estar preparados para la inclusión de nuevos usuarios y equipos a la red, adaptarlos a la red actual y asegurarse que tengan accesos a todos los recursos proveídos por ella; las redes actuales no proveen este tipo de escalabilidad, ya que el uso futuro de una red es generalmente impredecible[1],[2].

- **Políticas inconsistentes**: Implementar una política para toda la red supone tener que configurar miles de dispositivos y mecanismos. Generalmente estas redes requieren mucha experiencia para su administración, lo cual impide que el mantenimiento de las mismas sea más dinámico y acorde a las necesidades inmediatas de las empresas, para configurar una política de red, como una ACL o QoS, en una red estática grande toma mucho esfuerzo humano ir dispositivo en dispositivo introduciendo las configuraciones necesarias, incluyendo que esto también está sujeto al error humano que puede ocurrir al realizar tareas repetitivas tantas veces como sea necesario, esto exacerbado por la complejidad de las redes actuales, vuelve tarea difícil para los administradores de red introducir cambios, por pequeños que sean, a la configuración lógica de su red de datos[3].

- **Complejidad**: las redes actuales han crecido tanto que se han creado innumerables protocolos adaptados para situaciones específicas, lo cual ha causado que se incremente la complejidad al manejar una red de datos, con la inclusión de un nuevo dispositivo, se debe configurar y modificar las configuraciones de la gran parte de los dispositivos de red, creando una excesiva cantidad de trabajo para este tipo de tareas. Los departamentos de redes de las empresas prefieren mantener las redes de forma estáticas (es decir, no añadir nuevos equipos sólo en caso de ser necesario) para evitar al máximo esta complejidad y, por ende, evitar también la interrupción del servicio que viene con estas adiciones[4].

Este comportamiento de las redes no va acorde al comportamiento en el lado de los servidores, donde las tecnologías como virtualización han proveído de una gran flexibilidad al momento de la creación de nuevos recursos computacionales, así como optimizar al máximo el recurso físico disponible, creando una disparidad entre la flexibilidad de los servidores con la estaticidad de las redes de datos, lo que generalmente causa inconvenientes cuando se intentan integrar ambas en un espacio único[3].

1.1.1 Nuevos comportamientos en las redes de datos

Con la aparición de nuevos usos de la tecnología, se añaden nuevas maneras de tratar el tráfico de las redes, adaptándose a las necesidades de las empresas:

- **Cambio en los patrones de tráfico**: tiempo atrás, los datos que viajaban a través de las redes siempre fueron considerados generales, es decir, no existía clasificación en los mismos por servicios, sólo se implementaba seguridad a nivel de VLAN para agrupar las redes por usuarios y no por aplicaciones. En la actualidad, existe una mayor importancia en la clasificación de los datos que viajan por servicios o aplicación, ya que las prioridades en la disponibilidad de los datos se rigen por el tipo de servicio que proveen y su importancia para el negocio.

  Cuando las aplicaciones de red eran de tipo cliente-servidor, la información se obtenía de una sola fuente y sólo iba en una vía, donde el cliente consumía lo que el servidor enviaba; actualmente la información es solicitada por muchos usuarios al mismo tiempo, y así mismo, la información es proveída por muchos servidores, creando muchas vías de comunicación entre usuarios finales y servidores, que muy difícilmente pueden ser manejada con las herramientas actuales para la administración de las redes de datos[4].

- **Generalización de los servicios de información**: con la explosión de uso de teléfonos inteligentes, existen muchos más equipos que se conectan a la red administrada, por lo que es necesario proveer servicios rápidos y efectivos a estos nuevos dispositivos sin afectar a los usuarios actuales de la red.

- **Servicios en la nube**: con la nube, se agregan nuevas variables a los servicios prestados en una red, ya que pueden ser usados en nubes públicas, privadas o híbridas, donde la seguridad de la información es muy importante y la auditoría de la red es una medida rutinaria; este tipo de servicios requieren escalabilidad dinámica para poder crecer dependiendo de las necesidades.

La solución a todas estas problemáticas ocurre con el surgimiento de las Redes Definidas por Software (SDN), una arquitectura de red que separa el plano de control del plano de datos para conseguir redes más programables, automatizables y flexibles. Con SDN se virtualiza la red independizándola de la infraestructura física subyacente[5].

1.2. Beneficios de SDN

Una vez que el control del software es separado del hardware y transformado en un entorno más abierto, los usuarios pueden obtener diversos beneficios incluyendo los siguientes:

- ✓ **Visión unificada de la estructura de red**: Con el SDN se tiene una visión unificada de la red, simplificando la configuración, gestión y aprovisionamiento.
- ✓ **Reducción de costos:** Además de ser necesario un menor número de profesionales, el nivel de especialización también es reducido con la independencia en relación a los grandes proveedores. El costo total de propiedad (TCO) y los costos operacionales de la infraestructura también son reducidos, proporcionando más economía a los administradores. En los *Data Centers*, esa reducción se refleja directamente en los costos de interface, permitiendo la adopción de velocidades mayores en el acceso, como 40G *ethernet*. Este tipo de velocidades permiten la reducción en la cantidad de interfaces, lo que resulta en una reducción de costos de cableado y OPEX[6].
- ✓ **Alta utilización**: Una Ingeniería de tráfico centralizada proporciona una visión global, tanto en la oferta como en la demanda de recursos en la red. Gestionar a través de esta visión global el camino extremo a extremo que permita conseguir una alta utilización de los enlaces[1].
- ✓ **Manejo más rápido de fallos**: Los fallos, ya sean en un enlace o nodo, se gestionan más rápido. Además, los sistemas convergen más rápidamente hacia el objetivo óptimo y el comportamiento es predecible[5].
- ✓ **Disponibilidad, confiabilidad y seguridad:** Gracias a su capacidad de definición de políticas y reglas específicas, en un nivel bastante granular, las arquitecturas SDN pueden garantizar mayor disponibilidad, confiabilidad y seguridad del ambiente, ya que se elimina la necesidad de configuración manual e individual a cada adición o cambio de elementos de red, reduciendo el riesgo de fallas y consecuentes indisponibilidades[7].
- ✓ **Elasticidad en el cómputo**: Calcular la capacidad de los dispositivos de red ya no es un factor limitador, al igual que el control y la gestión, reside en los servidores y controladores externos.
- ✓ **Agilidad en el desarrollo de aplicaciones:** Una de las características más aclamadas por sus defensores es la rápida respuesta de las redes basadas en *software* a las

demandas de negocios. Con una configuración más simple y control centralizado, los administradores de red consiguen adecuar la infraestructura conforme la necesidad del usuario final. La virtualización del ambiente de red permite aún la definición de políticas de tráfico escalables y flexibles – basadas en la aplicación. La idea principal es permitir que alteraciones en las aplicaciones o nuevos despliegues se reflejen directamente en la capa de red[1],[7].

- ✓ **Actualizaciones sin impacto**: El desacoplamiento del plano de control respecto al plano de datos, permite llevar a cabo actualizaciones de software sin pérdida de paquetes o degradación de la capacidad[7].

1.3. Aplicaciones de SDN

Las redes SDN tienen aplicaciones en una gran variedad de entornos de red. Separando los planos de control y de datos, las redes programables permiten un control personalizado y una oportunidad para eliminar middleboxes y con ello simplificar el desarrollo y la implementación de nuevos servicios y protocolos. A continuación, se examinarán diferentes entornos para los que han sido propuestas o aplicadas soluciones SDN:

- ***Big Data:*** El gerenciamiento, almacenamiento y acceso a los datos corporativos viene cambiando significativamente en los últimos años. Al mismo tiempo en que el volumen de informaciones no-estructuradas aumentó exponencialmente, las arquitecturas de los Data Warehouses está cambiando, adaptándose a las nuevas necesidades y a las nuevas tecnologías propuestas por los proveedores. Esos cambios también demandan alteraciones en la arquitectura de las redes y en el standard de enrutamiento de los datos.
- ***Internet of Things:*** Las llamadas “Redes de sensores” y/o “Redes de comunicación entre máquinas (M2M)”, que darán origen a Internet de las Cosas (IoT, por sus siglas en inglés), exigen que la latencia de las redes sea la menor posible, la disponibilidad debe ser cercana a 100% y que los sistemas de seguridad no interfieran en las operaciones normal. En ese contexto, características como disponibilidad, confiabilidad, flexibilidad y seguridad son fundamentales[7].

- **Movilidad:** No resulta una novedad que los usuarios corporativos están demandando más movilidad, exigiendo la posibilidad de acceder a sistemas e informaciones en cualquier momento, en cualquier lugar y usando cualquier dispositivo, incluido su teléfono personal. Este cambio de comportamiento de los usuarios está alterando también los patrones de tráfico en las redes y en los Data Centers[4].
- ***Cloud Computing:*** Diferente del mundo cliente-servidor, en que la comunicación es bidireccional entre dos puntos, en el mundo Cloud las aplicaciones necesitan acceder a múltiples bancos de datos y servidores, generando tráfico en múltiples sentidos y ampliando la complejidad del ambiente. Entregar servicios de TI con agilidad y en el modelo "Self-Service" prepago por Cloud Computing exige que el procesamiento, almacenamiento y capacidad de red sean escalables[8].

1.4. Arquitectura de las SDN

- **Plano de infraestructura o de datos**: De manera similar a una red tradicional, el plano de infraestructura está compuesto por un conjunto de dispositivos, conmutadores, enrutadores interconectados entre sí y formando una gran red. La diferencia principal reside en el hecho de que los dispositivos físicos tradicionales son ahora elementos que se especializan en el reenvío de paquetes y no poseen un control o software embebido para tomar decisiones de forma autónoma. La inteligencia de la red ya no está en los dispositivos del plano de datos y pasa a un control lógicamente centralizado llamado NOS (Sistema Operativo de Red, por sus siglas en inglés)[9].

- **Plano de control**: Es una entidad lógica centralizada que tiene la función de convertir las solicitudes de las aplicaciones SDN en ordenes hacia los dispositivos de las capas inferiores y así proporcionarles a las aplicaciones SDN una visión abstracta de la red. Es el responsable de tomar la decisión sobre como los paquetes deberán ser reenviados por uno o más elementos de red, así como de la programación de los nodos de la red para implementar la decisión tomada. Tiene la función de mantener actualizada las tablas de reenvío de los dispositivos del plano datos o infraestructura basado en la topología de la red o solicitudes de servicios externos. Por medio de este

plano, se crea una vista centralizada de la topología de la red de datos, la cual permite gestionar el flujo de datos en el plano de infraestructura por medio de protocolos estándares, también brinda una API para que desde la capa de aplicación se puedan programar flujos y retroalimentar a la aplicación, con información de la topología de red o el tráfico de paquetes[8],[10].

- ✓ **Plano de aplicación**: Permite crear aplicaciones para automatizar tareas de configuración, provisión y despliegue de nuevos servicios en la red. Las aplicaciones de esta capa pueden considerarse el cerebro de la red debido a ellas implementan la lógica de control que se traduce en mandos que son enviados al plano de datos para dictar el comportamiento de los nodos especializados en el reenvío de paquetes. Las aplicaciones pueden definir la ruta a través de la cual los paquetes fluyen de un punto a otro de la red[11].

1.5. Protocolo OpenFlow

El protocolo OpenFlow fue originalmente propuesto como una alternativa para el desarrollo de protocolos experimentales en el campus de una universidad, en donde es posible probar nuevos algoritmos sin interrumpir o interferir con la normal operación del tráfico y surgió a raíz del proyecto de investigación "OpenFlow: *Enabling Innovation in Campus Networks*" en la universidad de Stanford en el 2008[12].

OpenFlow se define como un protocolo emergente y abierto de comunicaciones y es la primera interfaz de comunicaciones estándar definida entre los planos de control y datos de una arquitectura de SDN. OpenFlow permite el acceso directo y manipulación del plano de reenvío de los dispositivos de red tales como conmutadores y routers, tanto físicos como virtuales. OpenFlow hace posible que se pueda mover el control de la red fuera de los dispositivos para centralizarlo lógicamente en el software de control y establecer las primitivas básicas para programar el plano de reenvío.

El protocolo OpenFlow se implementa entre la infraestructura de los dispositivos de red y el software de control de SDN. OpenFlow utiliza el concepto de flujos para identificar el tráfico de red basado en reglas de coincidencia predefinidas que pueden ser programadas de forma

estática o dinámica por el un controlador SDN lo cual permite definir cómo el tráfico debe fluir a través de los dispositivos de red.

En la actualidad la estandarización de OpenFlow está a cargo de la *Open Networking Foundation* y lo hace a través de grupos de trabajo técnicos encargados de la configuración, pruebas de interoperabilidad, y otras actividades del protocolo, ayudando a garantizar la interoperabilidad entre los dispositivos de red y software de control de diferentes proveedores[6].

1.5.1. Funcionamiento de OpenFlow

OpenFlow aprovecha el hecho que las mayorías de los switches Ethernet contienen tablas de flujo (*Flow-Tables*), aunque cada una de estas son propia de los fabricantes, se han identificado varias características en común las cuales son utilizadas por OpenFlow para programar dichas tablas.

Cuando el primer paquete de un flujo llega al switch, este verifica las tablas de flujo con el objetivo de encontrar coincidencias, si ninguna coincidencia es encontrada el paquete es enviado al controlador y este es el encargado de insertar una entrada de flujo en la tabla del switch, luego de insertar esta entrada los paquetes que coincidan con la nueva entrada añadida se envían directamente a su destino sin la necesidad de enviarlo al controlador[6].

1.6. Controladores para SDN

Un controlador SDN ofrece una interfaz de programación para los conmutadores Openflow de tal forma que, las aplicaciones de gestión, a través de la misma, pueden realizar tareas de gestión y ofrecer nuevas funcionalidades. Un controlador SDN puede ser descrito de forma general como un sistema de software, o colección de sistemas, que ofrecen:

- Gestión del estado de la red, que implica una base de datos. Estas bases de datos sirven como un repositorio para la información de los elementos de red gestionados, incluyendo el estado de la red, alguna información de configuración temporal e información sobre la topología de la red[13].
- Un modelo de datos de alto nivel que captura las relaciones entre los recursos gestionados, las políticas y otros servicios prestados por el controlador. En muchos casos estos modelos de datos se construyen utilizando el lenguaje de modelado Yang.

- Un mecanismo de descubrimiento de dispositivos, topología y servicio; un sistema de cálculo de ruta y, potencialmente, otros servicios de información centrados en la red o en los recursos.
- Una sesión de control segura sobre el Protocolo de Control de Trasmisión (TCP) entre el controlador y los agentes asociados en los elementos de la red, por ejemplo, con el uso del protocolo TLS (Seguridad en la Capa de Transporte).[13]
- Un protocolo basado en estándares (OpenFlow) para obtener el estado de la red impulsado por las aplicaciones de los elementos de red.
- Un conjunto de APIs, a menudo RESTful (Transferencia de estado representacional) que exponen los servicios del controlador a las aplicaciones de gestión. Esto facilita la mayor parte de la interacción del controlador con estas aplicaciones. Esta interfaz se representa a partir del modelo de datos que describe los servicios y funciones del controlador. En algunos casos, el controlador y su API son parte de un entorno de desarrollo que genera el código de la API a partir del modelo de datos.
- Algunos controladores ofrecen entornos de desarrollo robustos que permiten la expansión de las capacidades básicas del núcleo y la posterior publicación de las APIs para los nuevos módulos, incluyendo los que soportan la expansión dinámica de las capacidades del controlador.
- Por tanto, en las SDN, es el controlador central el que dicta el comportamiento general de la red a partir de los requerimientos de las aplicaciones. Ejemplos de algunos controladores de código abierto existentes son: Beacon, Floodlight, NOX, POX, Ryu, Trema y OpenDayLight (ODL). En la figura 1.1 se muestran algunas características de estos controladores.

	Beacon	Floodlight	NOX	POX	Trema	Ryu	ODL
Soporte OpenFlow	OF v1.0	OF v1.0, 1.3, 1.4, 1.5	OF v1.0	OF v1.0, 1.3, 1.4, 1.5	OF v1.3	OF v1.0, v1.2, v1.3, v1.4, 1.5 y extensiones Nicira	OF v1.0
Virtualización	MiniNet y Open vSwitch	MiniNet y Open vSwitch	MiniNet y Open vSwitch	MiniNet y Open vSwitch	MiniNet y Open vSwitch	Construcción de una herramienta virtual de simulación	MiniNet y Open vSwitch
lenguaje de Desarrollo	Java	Java	C++	Python	Rudy/C	Python	Java
Provee REST API	NO	SI	NO	NO	Si (Básica)	Si (Básica)	SI
Interfaz Grafica	Web	Web	Python+, QT4	Python+, QT4 , WEB	NO	WEB	WEB
Soporte de plataformas	Linux, Mac OS, Windows y Android para móviles	Linux, Mac OS, Windows	Linux	Linux, Mac OS, Windows	Linux	Linux	Linux, Mac OS, Windows
Soporte de OpenStack	NO	SI	NO	NO	SI	SI	SI
Multiprocesos	SI	SI	SI	NO	SI	NO	SI
Tiempo en el mercado	5 años	4 años	8 años	3 años	4 años	4 años	3 años
Documentación	Buena	Buena	Media	Pobre	Pobre	Media	Media

Fig1.1 Características de algunos controladores. Fuente: Alejandro García Centeno,2014

1.6.1 Características de algunos controladores

A continuación, se hará una breve descripción de cada uno de los controladores más utilizados en las SDN, teniendo en cuenta parámetros como lenguaje de desarrollo, rendimiento, soporte de plataformas, interfaz gráfica, etc.

1.6.1.1 Controlador Beacon

Es un controlador OpenFlow de código abierto basado en Java creado en 2010. Se exploró nuevas áreas del espacio de diseño del controlador OpenFlow, con un enfoque en ser amistoso con el desarrollador, de alto rendimiento, y tener la capacidad para iniciar y detener aplicaciones nuevas o existentes en tiempo de ejecución. Beacon mostró sorprendentemente alto rendimiento, y fue capaz de escalar linealmente con núcleos de procesamiento, manejo de 12,8 millones de paquetes en mensajes por segundo con 12 núcleos, mientras que se construye utilizando Java. Beacon aprovecha múltiples librerías de las bibliotecas en la plataforma en un intento de maximizar la reutilización de código y para aliviar la carga de desarrollo tanto del controlador en sí mismo, y las aplicaciones de usuario. La biblioteca más importante es la Spring, proporciona un marco para el control de los dispositivos de red

utilizando el protocolo OpenFlow, y un conjunto de aplicaciones integradas que proporcionan funcionalidad de plano de control comúnmente necesario.[9],[14].

1.6.1.2 Controlador Floodlight

El controlador Floodlight es un controlador OpenFlow de clase empresarial, con licencia de Apache, basada en Java. Es apoyado por una comunidad de desarrolladores que incluyen una serie de ingenieros de redes de conmutación grandes, este trabaja entre el plano de datos y el plano de control.

Floodlight está diseñado para trabajar con el creciente número de switches, routers, switches virtuales, y los puntos de acceso compatibles con el estándar OpenFlow, a partir de Floodlight v1.0, Floodlight tiene soporte completo para todas las características OpenFlow 1.3. También es compatible con versiones anteriores de OpenFlow 1.0 (con soporte experimental para OpenFlow 1.1 y 1.2). Expone todas las versiones OpenFlow a través de una API común generado por el proyecto Loxigen llamado OpenFlowJ-Loxi. [15]

1.6.1.3 Controlador NOX

Se trata de una plataforma para la creación de aplicaciones de control de red. De hecho, mientras que lo que ahora se llama SDN creció de una serie de proyectos académicos (quizás principalmente SANE and Ethane), la primera tecnología SDN para obtener el reconocimiento el verdadero nombre era OpenFlow y NOX, fue desarrollado inicialmente en Nicira Redes de lado a lado con OpenFlow - NOX fue el primer controlador OpenFlow. Nicira donó NOX a la comunidad de investigadores en 2008, y desde entonces, ha sido la base de muchos y diversos proyectos de investigación en la exploración temprana del espacio SDN.[16]

1.6.1.4 Controlador POX

Es ideal para comenzar en SDN usando Python en Windows, Mac OS o Linux, desde el mismo instante que se instala este controlador con OpenFlow en sólo unos segundos después se puede utilizar su potencia. Está dirigido principalmente a la investigación y la educación, y es usado para el trabajo en curso sobre la definición de las abstracciones y técnicas clave para el diseño del controlador.

POX es el hermano menor de NOX. En su esencia, es una plataforma para el rápido desarrollo y creación de prototipos de software de control de red utilizando Python. Es decir, en un nivel muy básico, que es uno de un número creciente de framework, para ayudar a escribir un controlador OpenFlow.

POX también va más allá de esto. Además de ser un framework para interactuar con switches OpenFlow, se está usando como base para algunos trabajos en curso para ayudar a construir la disciplina emergente de *Software Defined Networking*. Se está usando para explorar y distribuir prototipos, SDN depuración, virtualización de redes, diseño del controlador, y modelos de programación. Soporta los mismos GUI y herramientas de visualización como NOX[17].

1.6.1.5 Controlador TREMA

Es un framework para programar un controlador SDN, a gusto del programador, en los lenguajes Ruby y C. Este entorno de programación genera un fichero de configuración que después puede ser ejecutado por Trema para comportarse como un controlador SDN.[2]

1.6.1.6 Controlador RYU

Es un componente framework para las redes definidas por software. Ryu ofrece componentes de software con API bien definidas que hacen que sea fácil para los desarrolladores crear nuevas aplicaciones de gestión y control de la red. Ryu es compatible con varios protocolos para la gestión de dispositivos de red, tales como OpenFlow, Netconf, DE-*config*, etc. Ryu es totalmente compatible con v1.0, 1.2, 1.3, 1.4 y Nicira Extensiones. Todo el código está disponible gratuitamente bajo la licencia Apache 2.0. Ryu tiene una impresionante colección de librerías, que van desde el apoyo a múltiples protocolos southbound y las operaciones de procesamiento de paquetes diferentes de red. Con respecto a los protocolos southbound, Ryu soporta DE-CONFIG (Gestión OpenFlow y Configuration Protocol), OVSDB (Protocolo de gestión de base de datos Openvswitch), NETCONF, X FLOW (NetFlow y sFlow) y otros protocolos de terceros. NetFlow es apoyado por Cisco y otros, y es específico para una IP. Los protocolos de muestreo de paquetes de soporte y agregación NetFlow y sFlow, que se utilizan principalmente para la medición del tráfico de red. [13]

1.6.1.7 Controlador OpenDayLight

Es una plataforma abierta para la programación de redes para permitir la SDN y NFV para redes en cualquier tamaño y escala. El segundo lanzamiento de la comunidad " *Helium*" viene con una nueva interfaz de usuario y un proceso de instalación mucho más simple y personalizable gracias a la utilización del contenedor Apache Karaf.

OpenDayLight es la combinación de componentes de software que incluye un controlador totalmente conectable, las interfaces, protocolos de *plug-ins* y aplicaciones. Con esta plataforma común tanto los clientes como los proveedores pueden innovar y colaborar con el fin de comercializar soluciones basadas en NFV y SDN.

En el comunicado de helio, la plataforma ha evolucionado OpenDayLight en otras áreas clave, como la alta disponibilidad y seguridad, así como el fortalecimiento y la adición de nuevos protocolos como OpenFlow, tipo de tabla Patrones, *PacketCable* Multimedia, y el framework de políticas de aplicaciones y herramientas para el servicio de función de encadenamiento[8].

OpenDayLight puede ser el componente central dentro de cualquier arquitectura SDN. Basándose en una fuente abierta SDN y el controlador NFV permite a los usuarios reducir la complejidad operativa, extender la vida de la infraestructura de hardware existente y permitir nuevos servicios y capacidades sólo disponibles con NEE. Ya sea que la organización es un proveedor de TI de la empresa, un proveedor de servicios de red o un proveedor de servicios en la nube, puede comenzar a tomar ventaja de SDN y NFV utilizando un controlador de código abierto impulsado por la comunidad disponibles en la actualidad[9].

1.7. Redes Inalámbricas Definidas por Software (SDWN)

Las Redes Inalámbricas Definidas por Software apuntan a proporcionar el control centralizado programático de la red fuera de las Puntos de Acceso inalámbricos (APs) ,los cuales ponen en vigor las instrucciones recibidas (decisiones de política) y permanecen responsables por la transmisión y recepción del trafico sobre los enlaces inalámbricos[19].

La separación de ambos planos de control y datos ha existido en el dominio inalámbrico antes que SDN y OpenFlow. IETF estandarizó el protocolo de Control de Aprovisionamiento de Puntos de Acceso Inalámbrico (CAPWAP) hace varios años, el cual centraliza el control en redes inalámbricas, permitiendo que los ACs (acces controllers) administren WTPs (Wireless

termination points) sobre una red inalámbrica. Además existen múltiples soluciones de propietarios (ej. Aerohive, Aruba, Cisco HDX,Meraki,Ruckus) basado en controladores externos responsables por la administración de los APs[20],[21].

Estas soluciones comerciales introducen un número de extensiones para protocolos estandarizados o para definir sus propias APIs entre el controlador y APs .Mientras todas estas soluciones han demostrado funcionar bien a la escala, su costo es a menudo prohibitivo para muchos despliegues y preocupante debido a la incapacidad para las innovaciones internas o de terceras aplicaciones[22].

SDWN se ha convertido en una emergente y significante rama de investigación de SDN, incluyendo la creciente atención de operadores de redes móviles y las identificadas sinergias a la Virtualización de las funciones de red (NFV) .Los emuladores de SDWN, por otro lado, serían una opción interesante para trabajar con los múltiples dispositivos (APs y STAs) a la escala razonable en los ambientes definidos experimentados, permitiendo investigación en nuevas funciones SDWN, además de los conocidos beneficios de la rica experimentación por la gran comunidad de usuarios de Mininet. Se provee que el emulador Mininet-WiFi tiene el potencial para convertirse en una importante herramienta para la investigación de SDN inalámbricas, habilitando los sistemas de redes inalámbricas del mundo real y sea un software (basado en Linux) de dispositivo de usuario final en un ambiente completamente controlado rindiendo resultados de alta fidelidad en apoyo a la investigación de SDWN[19],[23].

1.7.1 Introducción a SDWLAN

Durante los últimos años, el mundo de las telecomunicaciones ha experimentado un inesperado “boom” en el uso de los dispositivos inalámbricos, y un consecuente despliegue de redes inalámbricas extensivas en áreas tales como centros de negocios, aeropuertos, campus universitarios o incluso ciudades enteras. Esta situación ha dejado clara la necesidad de soluciones incluyendo un conjunto de Puntos de Acceso Inalámbricos (APs) coordinados, usualmente conocidos como “Enterprise Wi-Fi”. Aunque soluciones comerciales existen, estos son patentados, cerrados y costosos, las cuales en la mayoría de los casos las hace no viable para muchas organizaciones. En este contexto la comunidad científica está buscando

propuestas para soluciones de coordinación entre AP habilitando funciones avanzadas, ej: balanceo de carga, planificación de frecuencia o control de energía[24],[25].

La constante instalación, sea planificada o no, de los dispositivos para el acceso inalámbrico, originan una interferencia recíproca y, por tanto, la disminución del rendimiento de la red, convirtiéndose a la larga en un entorno inalámbrico inestable. Esta problemática sería mejorada con la implementación de una SDN) en la red de área local inalámbrica (WLAN = *Wireless Local Area Network*), la cual brinda la posibilidad de tener una red centralizada y flexible mediante la aplicación de distintas reglas a diferentes flujos de datos, permitiendo así gestionar numerosos puntos de acceso en una *WLAN* tradicional. La red inalámbrica de área local definida por software (SDWLAN = *Software Defined WLAN*) no sólo permite crear reglas en el flujo de datos, sino también controlar el comportamiento de los parámetros de los dispositivos inalámbricos (canales, control de acceso, potencia, etc.) [26],[27].

Una SDWLAN permite gestionar de forma flexible los dispositivos inalámbricos, empleando un controlador. Está demostrado que existe una mejora significativa en el rendimiento de una red al utilizar un controlador centralizado. Un controlador SDN puede obtener información de los dispositivos alrededor de cada punto de acceso que contengan en su firmware el protocolo OpenFlow, permitiendo conocer toda la topología de la red. Con esa información, es posible reducir la interferencia de los puntos de acceso adyacentes mediante el control de la potencia. Por otra parte, SDWLAN puede hacer que múltiples puntos de acceso actúen como un solo gran punto de acceso inalámbrico virtual, para evitar desconexiones de enlaces innecesarios[19],[27],[11].

CAPITULO 2. APLICACIONES DE SDN/NFV EN LAS WLAN.

En el presente capítulo se comenzará describiendo dos de las principales aplicaciones de SDN/NFV en las redes inalámbricas de área local (WLAN), una conocida como Punto de Acceso Virtual Ligero(LVAP) y otra como “ilusión de un gran AP”, luego se presentará una descripción concisa de las herramientas de simulación más utilizadas en las SDN. Por último, se presentará un breve tutorial de Mininet-WiFi, la herramienta por excelencia de las SDWLAN, donde se recogerán las principales funcionalidades que esta herramienta brinda.

2.1. Principales aplicaciones de SDN/NFV en las WLAN

2.1.1 Uso del Punto de Acceso Virtual Ligero (LVAP)

Uno de los principales problemas que surgen cuando se coordina una red inalámbrica es que las estaciones tienen sus propios algoritmos para seleccionar el AP. Por lo tanto, cada estación es libre de seleccionar el AP al cual asociarse, puramente en la base de decisiones locales, no coordinadas con el resto de los clientes. Esto complica la administración del cliente y resulta en problemas, ej:” el cliente pegajoso”, que nuca deja el AP al cual estuvo conectado inicialmente. Otro problema es que el *handoff* (transferencia desde un AP hacia otro) normal puede incurrir en una demora de varios cientos de milisegundos. Este hecho puede que no represente un gran problema para ciertos servicios, pero puede constituir una limitación severa para aplicaciones de tiempo real como VoIP o juegos en línea[28].

Una forma de solventar estos problemas es la introducción de la abstracción Punto de Acceso Virtual Ligero (LVAP). La idea es que un AP físico usara un LVAP diferente (el cual incluye una MAC especifica) para la comunicación con cada STA. Por lo tanto, la estación reconocerá un solo AP, aún si actualmente se está moviendo entre un conjunto de ellos, así evitando la necesidad de re-asociación.

Se puede decir que el LVAP viaja con la estación, se asigna a un AP físico cerca a la ubicación actual del terminal. Tanto como la estación vea solamente un simple AP, no se tomará ninguna decisión de roaming, así posibilitando a la red que soporte una administración coordinada de clientes. Esto es alcanzado sin ninguna modificación en la estación en la cual corre el estándar 802.11.

Varias soluciones han sido propuestas para el uso de LVAPs, entre ellos se propuso un protocolo para el intercambio de información entre APs. Una limitación de esta propuesta es que, debido a la ausencia de un controlador central, cada AP tiene que construir una lista de APs vecinos por sí mismo.

Otra solución basada en LVAPs consiste en combinar dos protocolos *Southbound:* OpenFlow y Odin. OpenFlow le dice a los switches internos de los APs donde dirigir el tráfico y el protocolo Odin está a cargo de los temas inalámbricos. El administrador central de la red Wi-Fi corre dentro del controlador SDN, así combinando ambas funcionalidades. El controlador está a cargo de crear un LVAP para cada terminal, el cual contiene cuatro campos: la MAC real de la estación, una MAC falsa para el AP para comunicarse con la estación, el IP de la estación y el SSID para ser usado en la comunicación[28].

Además, los algoritmos de administración de los recursos de radio corren como aplicaciones en la cima del controlador, habilitando una distribución óptima y dinámica de los STAs entre los APs.

Sin embargo, esta solución asume que todos los APs operan en el mismo canal, haciendo imposible desarrollar una adecuada planificación de frecuencias, el cual constituye una severa limitación para sus usos en despliegues reales. Finalmente, esta solución presenta una limitación de escalabilidad, como enviar tramas de broadcast no tiene sentido: cada LVAP tiene que trasmitir periódicamente tramas unicast a su estación usando la correspondiente MAC falsa del AP. *Handovers* rápidos y perfectos son parte esencial de todas aquellas soluciones porque las estaciones pueden ser redistribuidas dinámicamente. Por lo tanto, para no interrumpir la sesión de usuario, una reasignación de AP perfecta puede ser muy conveniente cuando el usuario está caminando, o siempre que una decisión de balance de carga se haga.

2.1.2 Ilusión de “Un gran AP”

El concepto de “Un gran AP” es una abstracción para clientes donde se prevé que todos los APs en WLAN empresariales colaborativamente ofrecen una ilusión para cada cliente, haciéndoles creer que hay un solo AP con una gran área de cobertura. Cuando el cliente se

mueve en el área de servicio, se equilibra la naturaleza de transmitir la señal inalámbrica para realizar un *handoff* de AP transparente al cliente y controlado de red[29].

Fig.2.1 Ilusión de un "Un gran AP". Se le hace creer al usuario que hay un solo AP con un área de cobertura muy grande y que siempre está conectado a este AP, aun cuando el AP asociado se ha cambiado. Fuente: (Dong Shao,2014)

En la ilusión "Un gran AP", cada cliente puede ver solamente un AP en el área de servicio, desde el punto de vista del cliente, este cree que siempre se está interactuando con un solo AP, mientras este se puede estar realmente comunicando con diferentes APs, en las WLAN empresariales el AP asociado al cliente juega dos roles: "receptor y retransmisor", y transmisor. En dirección de subida, AP es "receptor y retransmisor" que recibe los paquetes de cliente, responde ACKs a los clientes y envía paquetes al backbone alambrado que entrega paquetes a sus destinos. En dirección de bajada el AP es el transmisor, cada paquete destinado al cliente es primero entregado al AP asociado al cliente y luego envido al cliente.

La señal inalámbrica se transmite de forma tal que puede ser escuchada por los nodos cercanos. Por lo tanto, los paquetes de los clientes pueden ser recibidos por varios APs simultáneamente. La clave para lograr la ilusión de "Un gran AP" es un mecanismo de transferencia de AP rápida y transparente al cliente. La transferencia de AP debería ocurrir sin interrupción de tráfico continuo del cliente. Se puede lograr esto desde dos direcciones. En dirección de subida, se deja al AP viejo parar de recibir paquetes del cliente y responder ACK y dejar que el nievo AP reciba paquetes y responda ACK en nombre del viejo AP. En esta forma "receptor y retransmisor "del cliente cambiaron sin notificar al cliente. En dirección de bajada[29].

Se direccionan todos los paquetes al nuevo AP, y se deja al nuevo AP enviar paquetes al cliente. Para hacer creer al cliente que los paquetes provienen todavía del AP asociado, el

nuevo AP envía paquetes con la vieja BSSID. Aparentemente, se puede lograr un cambio silencioso del “receptor y retransmisor” y transmisor del cliente tanto como para asegurar que en cualquier momento hay uno y solo un AP proveyendo esos servicios para el cliente. Esto implica un alto grado de cooperación entre APs y el backbone cableado.

2.2. Herramientas de Simulación.

2.2.1 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 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 envían 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, muy parecidos a los switches y routers Ethernet reales. 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[30].

Mininet 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 maquina 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 una portátil, en un servidor, en una máquina virtual o en un sistema 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.
- Esta bajo un activo desarrollo y mejora continua.

Aunque Mininet aporta una 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, esto significa que las medidas estarán basadas en tiempo real y por lo tanto los resultados serán más rápidos que en la situación real[30].

2.2.2 Mininet-WiFi

Mininet-WiFi es una extensión de Mininet. Los desarrolladores de este aumentaron las funcionalidades de Mininet añadiendo estaciones (STAs) y Puntos de Acceso(APs) basados en el driver de dispositivos inalámbricos más común de Linux, Mac 802.11-SoftMac3.Esta extensión de Mininet tiene nuevas abstracciones y clases que soportan NICs inalámbricas y emulan enlaces conservando todas las virtualizaciones ligeras nativas y funciones OpenFlow/SDN, también se añaden atributos como la posición y el movimiento relativo de las estaciones móviles. Mininet-WiFi extiende el código base de Mininet, añadiendo, modificando clases y scripts. Por lo tanto, Mininet-WiFi añade nuevas funcionalidades y conserva aquellas que tenía Mininet[31],[32].

2.2.3 MiniEdit

MiniEdit se trata de un editor GUI para Mininet y Mininet-WiFi. 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.

2.2.4 Visual Network Description (VND*)*

VND es una plataforma web que tiene una interfaz gráfica para la creación de topologías de red SDN a ser utilizadas en la simulación y análisis de las mismas. VND permite realizar experimentos de SDN, generando scripts de Mininet en Python, los cuales están relacionados con el escenario de la red, las tablas de flujo y las configuraciones de QoS (*Quality of Service*). VND genera archivos compatibles con ns-3[33].

2.3. INTRODUCCIÓN AL USO DE MININET-WIFI.

2.3.1 Caracterización de Mininet-WiFi.

2.3.1.1 Mininet-WiFi y Movilidad.

Mientras el emulador de redes SDN Mininet puede ser usado para probar movilidad, Mininet-WiFi ofrece más opciones para emular escenarios complejos donde varias estaciones estarán cambiando los switches a los cuales están conectados. Mininet-WiFi añade nuevas clases que simplifican el trabajo de programación requerido por investigadores para crear escenarios de movilidad[34].

2.3.1.2 Emulación de WLAN .

Mininet-WiFi incorpora los drivers inalámbricos SoftMAC 802.11 de Linux, la interfaz de configuración inalámbrica cfg80211 y los drivers de simulación inalámbrica mac80211_hwsim en sus puntos de acceso.[24]

El driver mac80211_hwsim es un simulador para radios Wi-Fi. Este puede ser usado para crear interfaces virtuales Wi-Fi que usan el driver de LAN inalámbrica 802.11 SoftMAC. Usando esta herramienta investigadores pueden emular un enlace Wi-Fi entre máquinas virtuales. El driver 80211_hwsim les permite a los investigadores emular los mensajes de control de protocolo WiFi pasando entre puntos de acceso inalámbricos virtuales y estaciones móviles virtuales en un escenario de emulación de red. Por defecto ,80211_hwsim simula condiciones perfectas, las cuales significan que no hay pérdidas de paquetes o corrupción[2].

Se puede usar Wireshark para monitorizar el tráfico inalámbrico pasando entre los puntos de acceso inalámbricos y las estaciones inalámbricas virtuales en los escenarios de redes de Mininet-WiFi. Pero, resultará difícil capturar tráfico de control inalámbrico en las interfaces

estándares WLAN como ap1-wlan0 porque el kernel Linux quita los mensajes de control inalámbrico y cabeceras antes de hacer tráfico en estas interfaces disponibles para los procesos de usuario como Wireshark. Se tendrá que instalar herramientas adicionales y se seguirá un complejo procedimiento para habilitar el monitoreo del tráfico WiFi en la interfaz ap1-wlan0.Un método más fácil está disponible[25]:

Buscar la interfaz hwsim0 en el punto de acceso, habilitarla y monitorizar el tráfico en esta.

La interfaz hwsim0 repite las comunicaciones enviadas hacia las interfaces inalámbricas simuladas tales como ap1-wlan0 sin quitar ninguna cabecera 802.11 o control de tráfico.[34]

2.3.1.3 La gráfica de Mininet-WiFi.

Ya que las posiciones de los nodos es un aspecto importante en redes WiFi, Mininet-WiFi provee una exposición gráfica mostrando la ubicación de los nodos WiFi en una gráfica. La gráfica puede ser creada llamando este método en la API Python de Mininet-WiFi. Esta gráfica mostrará puntos de acceso inalámbrico, sus posiciones en el espacio y expondrá las afectaciones de los parámetros en el rango de cada nodo. La gráfica no mostrará ningún elemento de red cableado tales como los switches y estaciones de Mininet, conexiones Ethernet entre puntos de acceso, estaciones o switches[32].

2.3.2 Un Punto de Acceso

La red más simple es la topología por defecto, la cual consiste de un AP con dos estaciones inalámbricas. El Punto de Acceso es un switch conectado a un controlador. Las estaciones son hosts[35].

Este ejemplo demuestra como capturar tráfico de control inalámbrico y la forma en que un punto de acceso Open Flow maneja tráfico WiFi en una interfaz wlan.

Capturando tráfico de control inalámbrico en Mininet-WiFi.

Para ver tráfico de control inalámbrico primero se corre Wireshark:

```
wifi:~$ sudo wireshark &
```

Entonces se inicia Mininet-WiFi con el escenario de red por defecto usando el comando debajo:

```
wifi:~$ sudo mn --wifi
```

Luego, se habilita la interfaz hwsim0.La interfaz hwsim0 es la interfaz de software creada por Mininet-WiFi que copia todo el tráfico hacia todas las interfaces inalámbricas virtuales en el escenario de red. Esta es la forma más fácil para monitorear todos los paquetes en Mininet-WiFi[34].

```
mininet-wifi> sh ifconfig hwsim0 up
```

Ahora en Wireshark, se refrescan las interfaces y luego se comienza a capturar paquetes en la interfaz hwsim0.

Luego se envía un ping desde la estación (sta1) a la estación 2 (sta2)

```
mininet-wifi> sta1 ping sta2
```

En Wireshark se ven las tramas inalámbricas y los paquetes ICMP encapsulados en tramas inalámbricas pasando a través de la interfaz hwsim0.

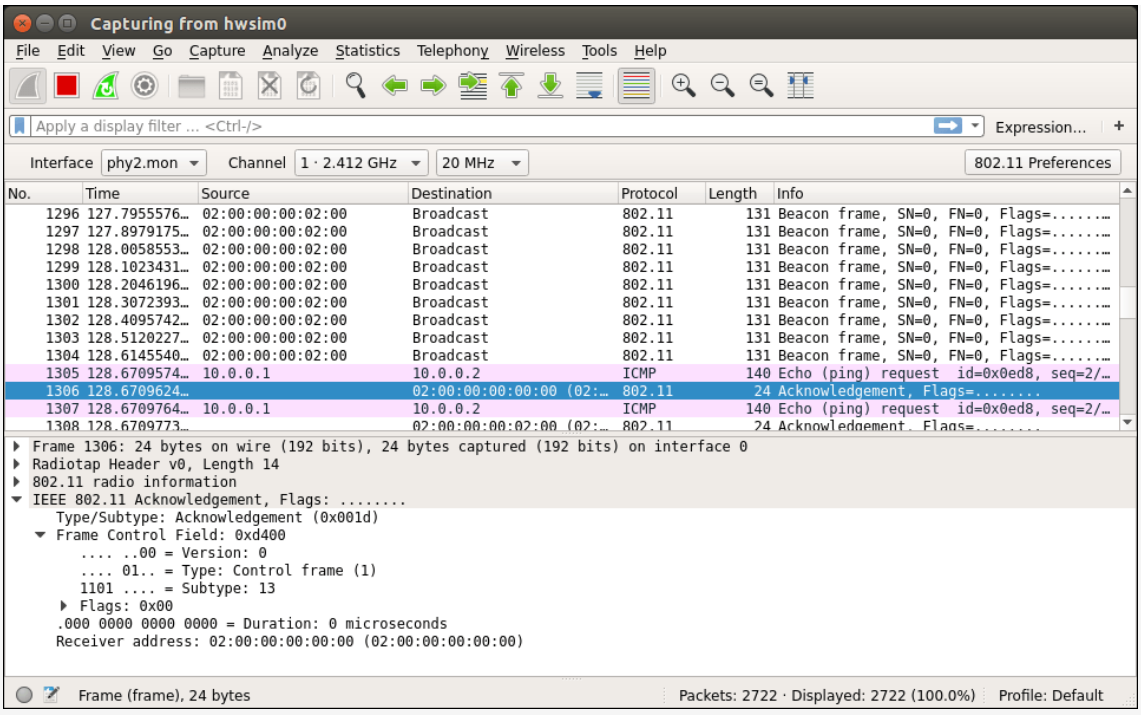

Figura 2.2 Captura de tramas inalámbricas pasando a través de la interfaz hwsim0. Fuente: Autor

Para comprobar si los flujos han sido creados:

```
dpctl dump-flows
```

2.3.3 Múltiples puntos de Acceso

Cuando se crea un escenario de red con dos o más puntos de acceso inalámbricos, se pueden apreciar más funciones disponibles en Mininet-WiFi. En esta parte se creará una topología linear con tres puntos de acceso, donde una estación es conectada a cada punto de acceso. Correr Mininet-WiFi y crear una topología lineal con tres puntos de acceso:

```
wifi:~$ sudo mn --wifi --topo linear,3
```

Desde la ejecución del comando se puede apreciar como la red es creada y cada estación es asociada con cada punto de acceso.

```
*** Creating network

*** Adding controller

*** Adding hosts and stations:

sta1 sta2 sta3

*** Adding switches and access point(s):

ap1 ap2 ap3

*** Adding links and associating station(s):

(ap2, ap1) (ap3, ap2) (sta1, ap1) (sta2, ap2) (sta3, ap3)

*** Starting controller(s)

c0

*** Starting switches and access points

ap1 ap2 ap3 ...

*** Starting CLI:

mininet-wifi>
```

Se puede verificar la configuración usando los comandos CLI de Mininet net y dump. Por ejemplo, se puede correr el comando net para ver las conexiones entre nodos:

```
mininet-wifi> net
```

```
sta1 sta1-wlan0:None

sta2 sta2-wlan0:None

sta3 sta3-wlan0:None

ap1 lo:  ap1-eth1:ap2-eth1

ap2 lo:  ap2-eth1:ap1-eth1 ap2-eth2:ap3-eth1

ap3 lo:  ap3-eth1: ap2-eth2

c0
```

Desde el comando net arriba, se puede ver que ap1, ap2, ap3 son conectados todos en una distribución lineal por enlaces Ethernet. Para chequear cuales puntos de acceso son visibles a cada estación, se usa el comando *iw*:

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan | grep ssid

        SSID: ssid_ap1

        SSID: ssid_ap2

        SSID: ssid_ap3
```

Para verificar el punto de acceso al cual está conectado cada estación actualmente se usa el comando de enlace iw. Por ejemplo, para ver el punto de acceso al cual está conectado la estación sta1 se usa el siguiente comando:

```
mininet-wifi> sta1 iw dev sta1-wlan0 link

Connected to 02:00:00:00:03:00 (on sta1-wlan0)

        SSID: ssid_ap1

        freq: 2412

        RX: 1853238 bytes (33672 packets)

        TX: 7871 bytes (174 packets)
```

```
        signal: -30 dBm
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi>
```

2.3.3.1 Un escenario de movilidad simple

En este ejemplo, cada estación está conectada a diferentes puntos de acceso inalámbricos. Se puede usar el comando *iw* para cambiar cada punto de acceso al que está conectada cada estación. Si se quisiera cambiar la asociación de sta1, el cual está asociado con ap1, a ap2:

```
mininet-wifi> sta1 iw dev sta1-wlan0 disconnect
mininet-wifi> sta1 iw dev sta1-wlan0 connect ssid_ap2
```

Se verifica el cambio con el comando de enlace *iw*:

```
Mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:04:00 (on sta1-wlan0)
        SSID: ssid_ap2
        freq: 2412
        RX: 112 bytes (4 packets)
        TX: 103 bytes (2 packets)
        signal: -30 dBm
        tx bitrate: 1.0 MBit/s
```

```
        bss flags:      short-slot-time

        dtim period:    2

        beacon int:     100

mininet-wifi>
```

Se aprecia que la estación sta1 está ahora asociada con ap2.

2.3.3.2 Flujo OpenFlow en escenarios de movilidad

Ahora se muestra como 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 que se conocen las direcciones IP de las estaciones se va a abrir una ventana externa para una de las estaciones:

```
mininet-wifi> xterm sta3
```

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

```
root@mininet-wifi:~# ping 10.0.0.1
```

Ahora se correrá 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:

```
mininet-wifi> 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 esta es la que se había modificado. Se debe tener en cuenta que al hacer esto 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:

```
mininet-wifi> dpctl del-flows
```

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

2.3.4 API Python y scripts

Mininet provee una API Python con la cual los usuarios pueden crear scripts Python que crean las topologías personalizadas. Mininet-WiFi extiende esta API que soporta un ambiente inalámbrico. Cuando se usa el comando de Mininet *mn* con la opción *–wifi* para crear topologías de Mininet-WiFi, no se tiene acceso a las funcionalidades más extendidas proveídas en Mininet-WiFi. Para acceder a las funciones que permiten emular el comportamiento de nodos en una LAN inalámbrica, se necesitan las extensiones de Mininet-WiFi para la API Python. Los desarrolladores de Mininet-WiFi añadieron nuevas funciones a Mininet que soportan emulación de nodos en un ambiente inalámbrico. Mininet-WiFi añade los métodos addStation y *addBaseStation* y un método *modifiedaddLink* para definir el ambiente inalámbrico. Existen ejemplos de scripts que muestran cómo usar la mayoría de las funciones en Mininet-WiFi. Los ejemplos de scripts de Mininet-WiFi están en el directorio ~/Mininet-wifi/examples.

2.3.4.1 Estación Base y métodos de punto de acceso

En un escenario simple, se puede añadir una estación y un punto de acceso con los siguientes métodos en un script Python de Mininet-WiFi:

Añadir una nueva estación llamada sta1, con todos los parámetros puestos a valores por defecto:

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

Añadir un nuevo punto de acceso llamado ap1, con SSID ap1-ssid, y todos los otros parámetros puestos a valores por defecto:

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

Añadir una asociación inalámbrica entre estación y punto de acceso, con valores por defecto para atributos de enlace:

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

Para escenarios más complicados, más parámetros están disponibles para cada método. Se puede especificar la dirección MAC, dirección IP, ubicación en el espacio tridimensional, rango del radio y más. Por ejemplo, el siguiente código define un punto de acceso y una estación, y crea una asociación (una conexión inalámbrica) entre los dos nodos y aplica algunos parámetros de control de tráfico, añadiendo restricciones de ancho de banda, una tasa de error, y una demora de propagación:

Añadir una estación y verificar el método de encriptación inalámbrico, la dirección MAC de la estación, dirección IP, y posición en el espacio virtual:

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

Añadir un punto de acceso y especificar el método de encriptación inalámbrico, SSID, modo inalámbrico, canal, posición, y rango de radio:

```
net.addBaseStation( 'ap1', passwd='123456789a', encrypt='wpa2', ss
id= 'ap1-ssid', mode= 'g', channel= '1', position='30,30,0', range
=30 )
```

Añadir una asociación inalámbrica entre una estación y un punto de acceso y especificar propiedades de enlace de ancho de banda máximo, tasa de error, y retardo:

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

Para activar el control de asociación en una red estática, se puede usar el método *associationControl*, el cual hace que Mininet-WiFi automáticamente escoja para una estación base a cuál punto de acceso esta se conectara basándose en el rango entre estaciones y puntos de acceso. Por ejemplo, usar el siguiente método para usar la primera señal más fuerte cuando se determine conexiones entre estaciones y puntos de acceso:

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

2.3.4.2 API Clásica de Mininet

La API Python de Mininet-WiFi todavía soporta los tipos de nodos estándares de Mininet-switches, hosts y controladores. Por ejemplo: Para añadir un *host* hay que tener presente que este *host* tiene una interfaz Ethernet a diferencia de las *stations* que tienen interfaces inalámbricas.

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

Añadir un switch:

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

Añadir un enlace Ethernet entre dos nodos. Note que si se usa addLink para conectar dos puntos de acceso juntos (y se está usando el modo infraestructura por defecto), Mininet-WiFi crea un enlace Ethernet entre ellos.

```
net.addLink( s1, h1 )
```

Añadir un controlador:

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

Usando la API Python, se puede construir una topología que incluye hosts, switches, estaciones, puntos de acceso, y múltiples controladores.

2.3.4.3 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' )
```

2.3.4.4 Trabajando con Mininet-WiFi durante la ejecución

Los scripts pueden ser ejecutados tanto corriendo directamente el script:

```
wifi:~/scripts $ sudo ./xxxxxxx.py
```

Como siendo parte de un comando de Python:

```
wifi:~$ 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 instrucciones 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 intérprete 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:

```
mininet-wifi> position ap1
```

- La distancia:

```
mininet-wifi> distance ap1 sta2
```

- Información (número de asociaciones, potencia, SSID)

```
mininet-wifi> info ap1
```

Intérprete de Python para la recolección de datos

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

```
mininet-wifi> py ap1.range
```

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

```
mininet-wifi> py ap1.associatedStations
```

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

```
mininet-wifi> py ap1.nAssociatedStations
```

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

```
mininet-wifi> 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.

2.3.4.5 Realización de cambios durante la ejecución.

Para cambiar el punto de acceso al que está conectado:

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

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

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

Para cambiar el rango:

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

2.3.5 Movilidad

La característica más importante que ofrece Mininet-WiFi respecto a Mininet es el soporte sobre la movilidad de las estaciones. En esta parte, se puede ver cómo manejar y crear un escenario donde las estaciones se muevan a lo largo del espacio y van cambiando de puntos de acceso a los que se conectan, basándose en la cercanía[36].

2.3.5.1 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( start ime=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)
```

En este ejemplo se puede observar un parámetro AC que es el encargado de las asociaciones de control:

- *iif (Least-Loaded-First)*

- *ssf (Strongest-Signal-First)*

2.3.5.2 Test con iperf

Para ver cómo responde el sistema al tráfico se realizará un *ping*, cuando ya se haya realizado ese *ping*. Se inicia un servidor *iperf* en la estación[35]:

```
mininet-wifi> sta1 iperf --server &
```

Tras esto, se abrirá una ventana de comando en un host o en una estación con el comando *xterm* ya mencionado.

```
mininet-wifi> xterm h1
```

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
```

Cuando se 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[34]:

```
mininet-wifi> dpctl del-flows
```

CAPITULO 3. SIMULACIÓN DE ESCENARIOS DE PRUEBA MEDIANTE MININET-WIFI.

En el presente capítulo se realizarán varias pruebas en 4 escenarios diferentes mediante la herramienta de simulación, previamente descrita, Mininet-WiFi, con el propósito de ver cómo se comportan las SDWLAN, se analizarán varios parámetros tanto en condiciones de estaticidad como en movilidad y se hará uso de un controlador remoto en cada uno de los ejemplos.

3.1. Controlador seleccionado para el desarrollo del proyecto

Saber elegir o seleccionar un controlador es muy importante en el diseño de una red SDN, ya que no todos cumplen con los mismos objetivos o están limitados a tareas muy específicas. Para este proyecto se escogió POX, el cual soporta distintas versiones de OpenFlow, cuenta con una interfaz OpenFlow para python y está destinado principalmente a la investigación y educación, además de su fácil implementación y uso, POX provee de componentes ya ejecutables y programados por sus desarrolladores en python. Los componentes que se pueden citar son los siguientes:

- *forwarding.l2_learning*: Éste complemento hace al switch OpenFlow manejarse como un switch de capa 2. Hace que el switch aprenda direcciones, además los flujos que se aplican son exactamente coincidentes para la mayoría de campos.

- *forwarding.l3_learning*: Complemento que es parecido con el anterior, salvo que éste actúa y posee funcionalidades como el switch de capa 3. Aprende y escucha sobre las IP que encuentra y maneja peticiones ARP (Protocolo de Resolución de Direcciones).

- *samples.spanning_tree*: Gracias a este componente se puede crear una vista de la topología de red, construye un *spanning tree*, y desconfigura el flujo en los puertos del switch que no están en la topología *spanning tree.*

- *Web.webcore:* Este componente inicia un servidor web con un proceso POX. Otros componentes pueden ser los que interactúen con este componente *web. web.core* para que provean contenido dinámico o estático.

- *Messenger*: Este componente proporciona una interfaz para que POX funcione con procesos externos, tratada como una API ya que la comunicación se la emplea por otros componentes llamados de transporte. Este componente está libre de uso para TCP y HTTP.
- *Openflow.of_01*: Este componente permite la comunicación entre versiones de switches Openflow 1.0.
- *Openflow.discovery:* Componente que envía mensajes LLDP (detección de nodos en una red) programados hacia los switches Openflow para poder revelar la topología actual de la red.

- *Openflow.debug*: Este componente hace que POX realice búsquedas en los mensajes Openflow.
- *Openflow.keepalive*: Componente que fuerza a POX a enviar solicitud *'echo'* a los switch conectados para ver el tiempo de demora de respuesta por parte de los switch.
- *Proto.dhcp_client:* Es un componente de cliente DHCP. No es tan útil por sí mismo, pero en conjunto con otros componentes puede llegar a ser de utilidad.
- *Proto.pong*: El componente *pong* es ineficiente ya que solo mira las solicitudes ICMP (al hacer ping) y los responde.

Para el proyecto se seleccionó el componente *forwarding. l2_learning*, ya que es el que más se ajusta a los requerimientos de las redes que se diseñarán. En los anexos III y IV se explican los pasos para la instalación y ejecución de POX.

3.2. Escenario de prueba 1: Estaciones fijas inalámbricas con autentificación

Para la realización de esta topología se utilizó el editor de Python que viene incluido en el sistema operativo Ubuntu 16.04, *gedit* es el editor de texto libre oficial del escritorio Gnome. Mininet-WiFi permite exportar las API de Python para crear redes personalizadas y utilizando el comando *python /xxxx.py* se pueden ejecutar las topologías creadas En la figura 3.1 se puede observar la topología de este escenario y en la figura 3.2 el código utilizado para esta topología.

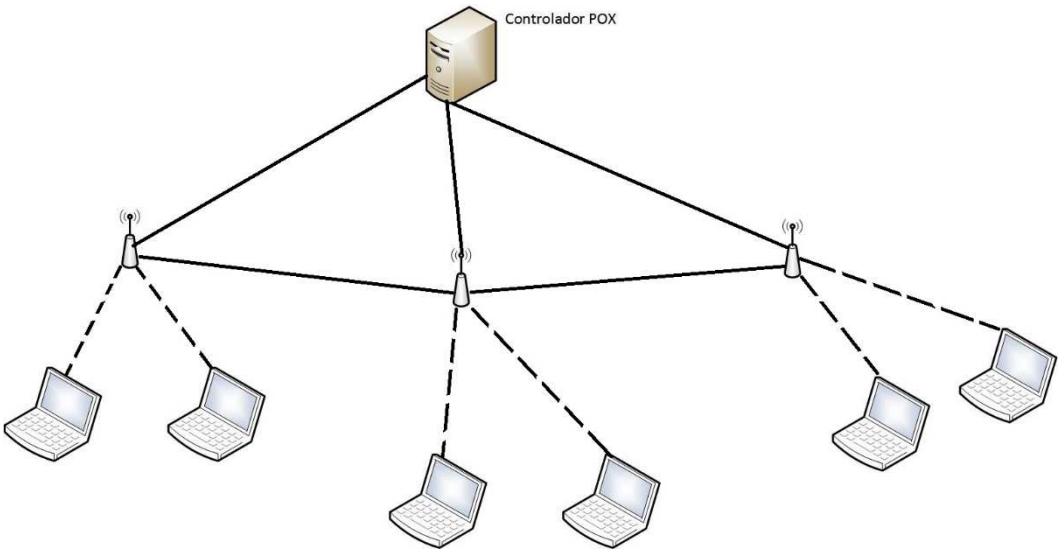

Fig. 3.1 Topología de red del primer escenario de pruebas. Fuente: Autor.

```
#!/usr/bin/python

'autor:Ramon Estrada de la Torre'

from mininet.net import Mininet
from mininet.node import  Controller,RemoteController, OVSKernelAP,OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel,info
from mininet.link import TCLink


def topology():
    "Create a network."
    net = Mininet(controller=RemoteController, link=TCLink, accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1',ip="10.0.0.1/24",passwd='123456789a', encrypt='wpa2')  # encrypt=(wpa,wpa2,wep)
    sta2 = net.addStation('sta2', ip="10.0.0.2/24",passwd='123456789a', encrypt='wpa2')  # encrypt=(wpa,wpa2,wep)
    sta3 = net.addStation('sta3',ip="10.0.0.3/24",passwd='123456789b', encrypt='wpa2')  # encrypt=(wpa,wpa2,wep)
    sta4 = net.addStation('sta4',ip="10.0.0.4/24",passwd='123456789b', encrypt='wpa2')  # encrypt=(wpa,wpa2,wep)
    sta5 = net.addStation('sta5',ip="10.0.0.5/24",passwd='123456789c', encrypt='wpa2')  # encrypt=(wpa,wpa2,wep)
    sta6 = net.addStation('sta6',ip="10.0.0.6/24",passwd='123456789c', encrypt='wpa2')  # encrypt=(wpa,wpa2,wep)
    ap1 = net.addAccessPoint('ap1', ssid="wifi1", mode="g", channel="1", passwd='123456789a', encrypt='wpa2')  #
encrypt=(wpa,wpa2,wep)
    ap2 = net.addAccessPoint('ap2', ssid="wifi2", mode="b", channel="6", passwd='123456789b', encrypt='wpa2')  #
encrypt=(wpa,wpa2,wep)
    ap3 = net.addAccessPoint('ap3', ssid="wifi3", mode="g", channel="11", passwd='123456789c', encrypt='wpa2')
# encrypt=(wpa,wpa2,wep)
    print "***Configurando controlador remoto"
    c0 = net.addController('c0', controller=RemoteController,ip="192.168.6.3",port=6633)

    print "*** Configurando nodos wifi"
    net.configureWifiNodes()

    print "*** Asociando estaciones"
    net.addLink(ap1,ap2)
    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])

    print "*** Running CLI"
    CLI(net)


    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Fig. 3.2 Código Python de Topología de red del primer escenario de prueba. **Fuente: Autor.**

En el código representado se tiene lo siguiente:3 Puntos de Acceso y 6 estaciones, las estaciones se reparten por los APs, el ap1 tiene conectado las estaciones sta1 y sta2, el ap2 tiene conectado sta3 y sta4 y ap3 tiene conectado sta5 y sta6. Lo primero en el código es importar las librerías de Mininet para crear el API, luego se instancia un objeto Mininet como se aprecia en el código, después se crean las estaciones, los APs y el controlador que en este caso va a ser un controlador remoto (Ver anexo IV), se agregan las relaciones entre ellos y finalmente se inicializan todos los dispositivos. En este caso a las estaciones se le introducen los siguientes parámetros: dirección IP, protocolo de encriptación WPA-2 y contraseña para autentificarse con el correspondiente AP ya que tanto las estaciones como los AP van a utilizar el protocolo de encriptación WPA-2. A los APs se le introducen los siguientes parámetros: SSID, modo, canal, protocolo de encriptación y contraseña.

Al ejecutar el archivo topo1.py se crea en la máquina virtual las interfaces virtuales y al mismo tiempo se compila el código de la emulación que se ha desarrollado. En la figura 3.3 se muestra como se realiza el llamado de la topología, se puede apreciar cómo se añaden y activan todos los dispositivos incluyendo el controlador remoto.

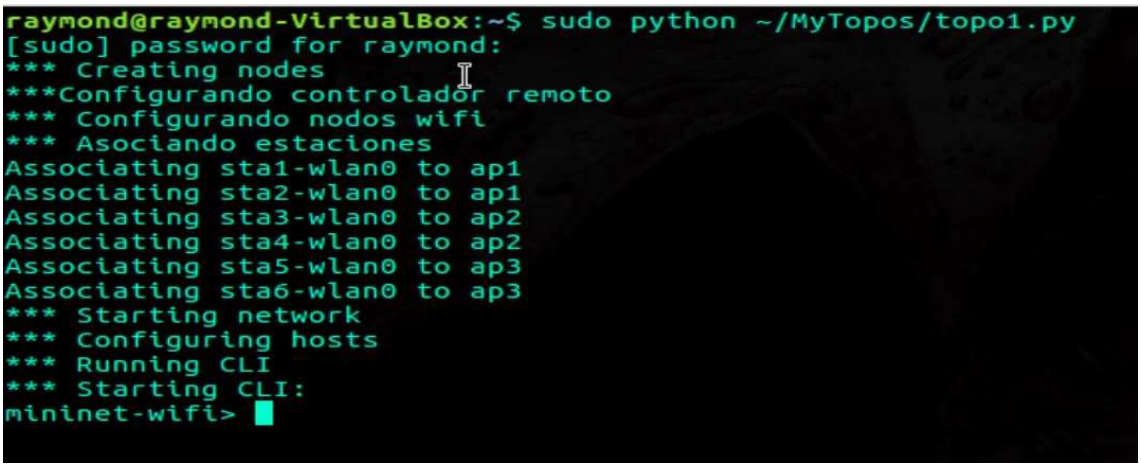

Fig.3.3 Llamado en Mininet-WiFi del código python. **Fuente: Autor.**

Una vez creada la red se procede a comprobar dicho funcionamiento haciendo un ping a todos los nodos mediante el comando *pingall*, como se puede observar en el fragmento de abajo hubo respuesta entre todos los dispositivos sin perdidas de paquetes.

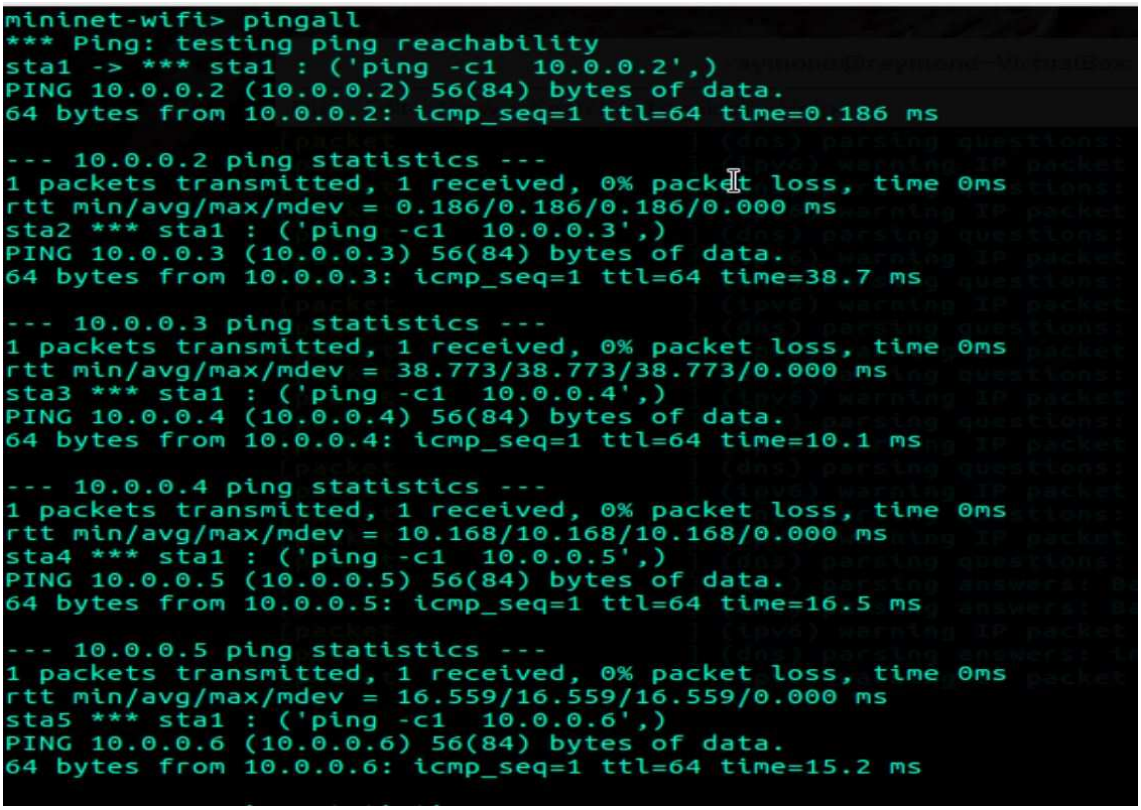

Fig.3.4 Verificación de conectividad de red. **Fuente: Autor.**

Para verificar los enlaces que tiene la red se utiliza el comando net, con esto solo se puede ver las conexiones de los puntos de acceso, ya que como las estaciones están conectadas de forma inalámbrica sería necesario utilizar otro comando.

```
mininet-wifi> net
sta1 sta1-wlan0:wireless
sta2 sta2-wlan0:wireless
sta3 sta3-wlan0:wireless
sta4 sta4-wlan0:wireless
sta5 sta5-wlan0:wireless
sta6 sta6-wlan0:wireless
ap1 lo:  ap1-wlan1:wireless ap1-eth2:ap2-eth2
ap2 lo:  ap2-wlan1:wireless ap2-eth2:ap1-eth2 ap2-eth3:ap3-eth2
ap3 lo:  ap3-wlan1:wireless ap3-eth2:ap2-eth3
c0
mininet-wifi>
```

Fig.3.5 Conexiones de red del primer escenario de red. **Fuente: Autor.**

Con el comando *sta1 iw dev sta1-wlan0 link* se verifica las conexiones de cada estación, en este caso de la estación sta1, para ir comprobando la conexión de las demás estaciones habría que ir cambiando por cada estación como se muestra en la figura 3.6.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:06:00 (on sta1-wlan0)
        SSID: wifi1
        freq: 2412
        RX: 1076562 bytes (15978 packets)
        TX: 4974 bytes (57 packets)
        signal: -30 dBm
        tx bitrate: 1.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi> sta2 iw dev sta2-wlan0 link
Connected to 02:00:00:00:06:00 (on sta2-wlan0)
        SSID: wifi1
        freq: 2412
        RX: 1099363 bytes (16322 packets)
        TX: 5049 bytes (58 packets)
        signal: -30 dBm
        tx bitrate: 1.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi> sta3 iw dev sta3-wlan0 link
Connected to 02:00:00:00:07:00 (on sta3-wlan0)
        SSID: wifi2
        freq: 2437
        RX: 1036563 bytes (17031 packets)
        TX: 5114 bytes (59 packets)
        signal: -30 dBm
        tx bitrate: 1.0 MBit/s

        bss flags:
        dtim period:    2
        beacon int:     100
mininet-wifi>
```

Fig.3.6 Verificación de la conexión de las estaciones del primer escenario de red. **Fuente: Autor.**

Próximamente se procede a comprobar las tablas de flujo creadas por el controlador en los nodos de la red, para ello se emplea el comando *dpctl dump-flows* y en la figura 3.7 se puede apreciar las tablas de flujo de los Puntos de Acceso ap1, ap2 y ap3.

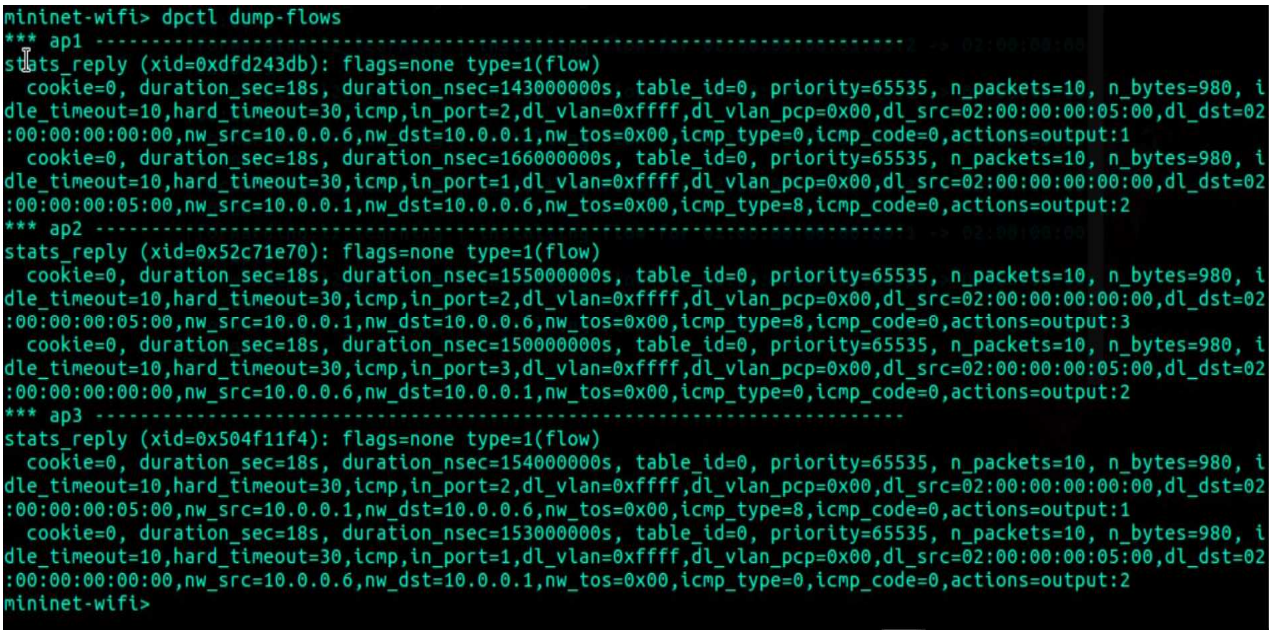

Fig.3.7 Tablas de flujo. **Fuente: Autor.**

A continuación, se realizará una serie de capturas, utilizando el analizador de protocolos Wireshark, para observar el comportamiento de este escenario a otro nivel, observando el tráfico OpenFlow y el tráfico de datos en este escenario, para ello se realizará un ping entre sta1 y sta5. Se capturará en diferentes interfaces, comenzando por la de *loopback*, luego en la *ap1-wlan1* y por último en la *hwsim0.*

Se habilita Wireshark a capturar en la interfaz de loopback y se obtendrá lo siguiente:

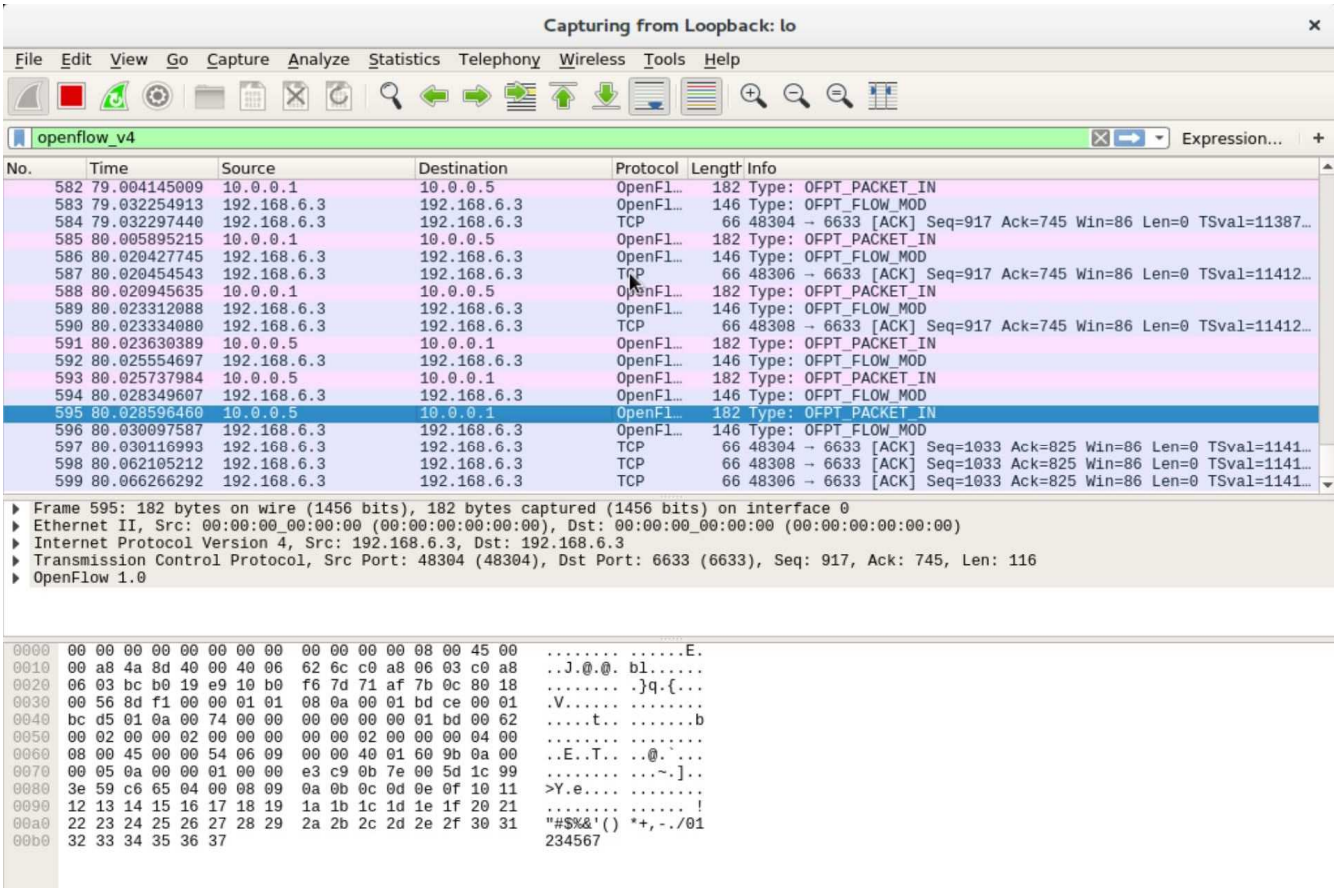

Fig.3.8 Captura de tráfico en la interfaz de loopback. **Fuente: Autor.**

En esta captura se pueden ver las peticiones de estadísticas y el protocolo OpenFlow.

En la siguiente figura se observa la captura en la interfaz de uno de los nodos: ap1-wlan1, donde se obtienen paquetes del protocolo ICMP, ARP y LLDP.

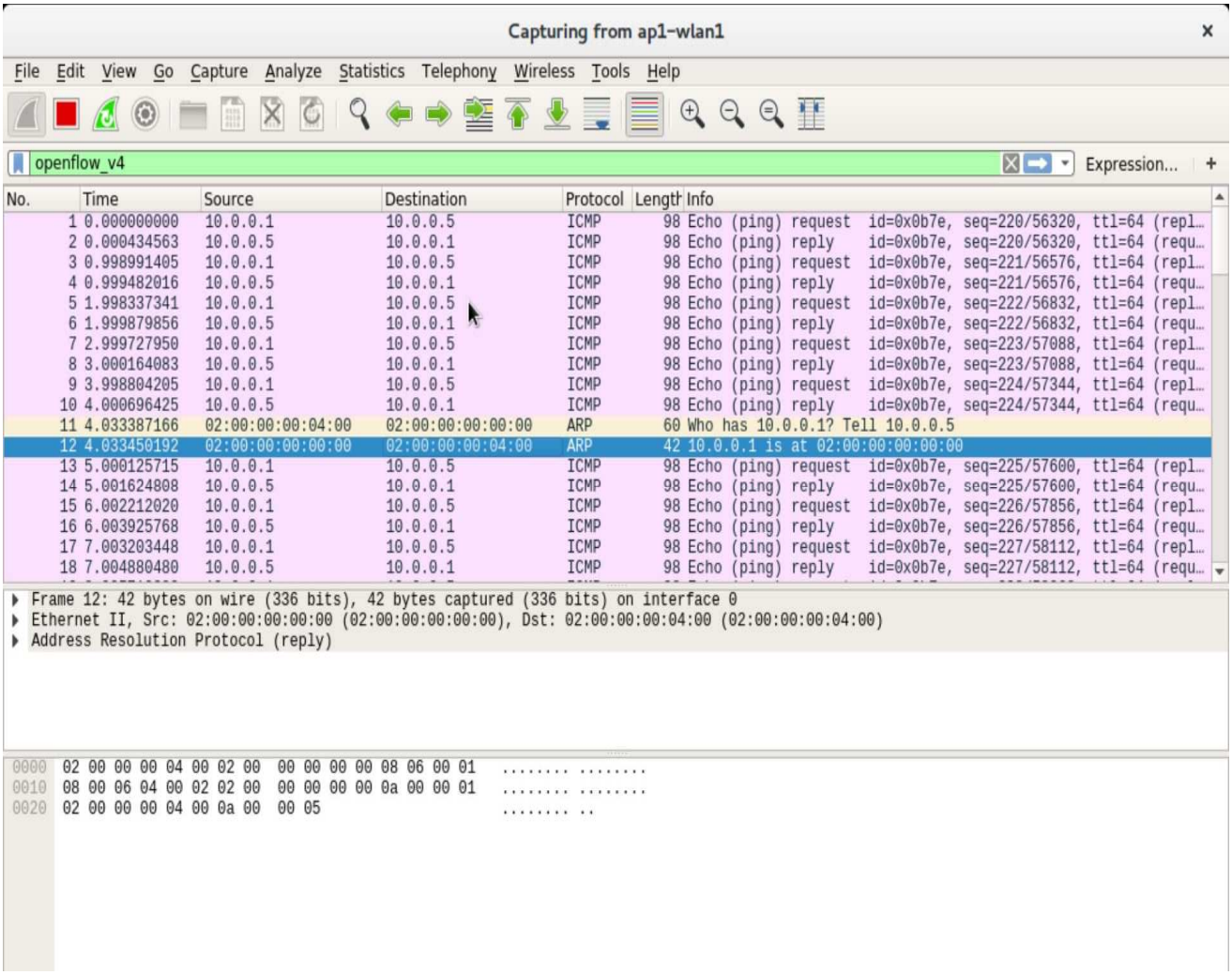

Fig.3.9 Captura de tráfico en la interfaz ap1-wlan1. **Fuente: Autor.**

Por último, se realiza la captura en la interfaz *hwsim0*, pero previamente hay que activarla con el siguiente comando *sh ifconfig hwsim0 up*

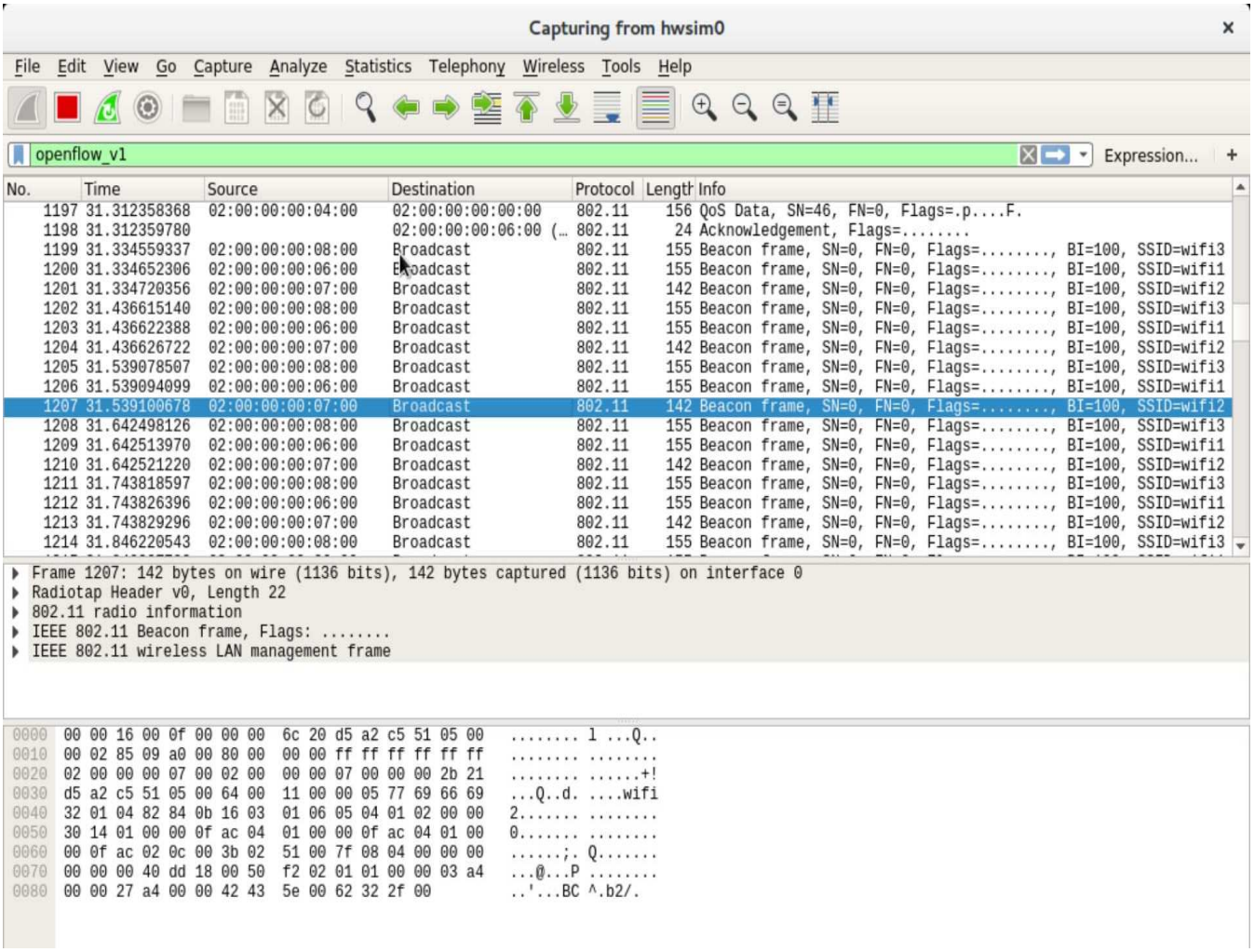

Fig.3.10 Captura de tráfico en la interfaz hwsim0. **Fuente: Autor.**

En esta captura se observan las tramas IEEE 802.11, ya que se trata de una conexión inalámbrica la de este escenario, y mediante esta interfaz es como mejor se observan este tipo de tramas.

3.3. Escenario de prueba2: Red Mesh Simple

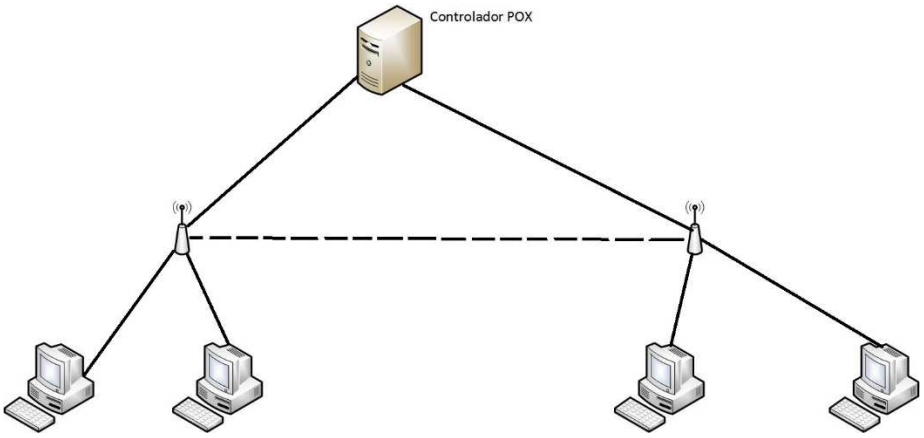

Fig. 3.11 Topología de red del segundo escenario de pruebas. Fuente: Autor.

```
#!/usr/bin/python

'Autor:Ramon Estrada de la Torre'

from mininet.net import Mininet
from mininet.node import Controller, RemoteController ,UserAP,OVSKernelAP
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.link import TCLink

def topology():
    "Create a network."
    net = Mininet(controller=RemoteController, link=TCLink, accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    h1 = net.addHost('h1', mac='00:00:00:00:00:11')
    h2 = net.addHost('h2', mac='00:00:00:00:00:12')
    h3 = net.addHost('h3', mac='00:00:00:00:00:13')
    h4 = net.addHost('h4', mac='00:00:00:00:00:14')
    ap1 = net.addWirelessMeshAP('ap1', ssid='ssid-ap1', mode='g', channel='6')
    ap2 = net.addWirelessMeshAP('ap2', ssid='ssid-ap2', mode='g', channel='6')
    c1 = net.addController('c0', RemoteController,ip="192.168.6.3",port=6633)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    print "*** Associating Stations"
    net.addLink(h1, ap1)
    net.addLink(h2, ap1)
    net.addLink(h3 ,ap2)
    net.addLink(h4 ,ap2)
    net.addMesh(ap1, ssid='RED-MESH')
    net.addMesh(ap2, ssid='RED-MESH')
```

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


    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Fig.3.12 Código Python de Topología de red del segundo escenario de prueba. Fuente: Autor.

Esta topología consiste en el funcionamiento básico de una simple red *mesh* con dos Puntos de Acceso conectados inalámbricamente y 4 hosts asociados alambradamente a los Puntos de Acceso, repartidos dos por cada AP. Como se puede apreciar se utiliza un controlador remoto al igual que en el escenario previo.

Una vez llamada la topología en Mininet-WiFi y creada la red se comprobará su funcionamiento haciendo un ping entre los diferentes hosts.

```
mininet-wifi> h1 ping -c1 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=54.4 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.489/54.489/54.489/0.000 ms
mininet-wifi> h2 ping -c1 h4
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=15.5 ms

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.547/15.547/15.547/0.000 ms
mininet-wifi>
```

Fig.3.13 Prueba de conectividad de la red del segundo escenario. Fuente: Autor.

En la figura se aprecian los pings realizados, comprobándose el correcto funcionamiento de la red.

3.4. Escenario de Prueba3: Estaciones fijas inalámbricas con control de asociación

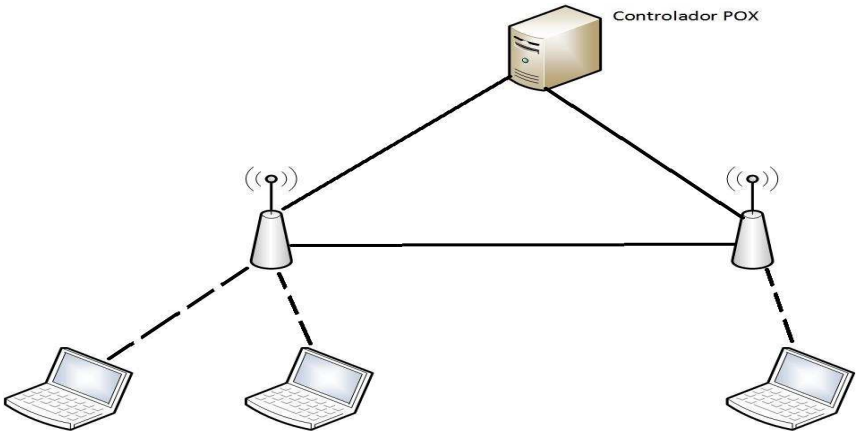

Fig. 3.14 Topología de red del tercer escenario de pruebas. Fuente: Autor

```
#!/usr/bin/pytho
'Autor:Ramon Estrada de la Torre'

from mininet.net import Mininet
from mininet.node import Controller,RemoteController, OVSKernelAP
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel

def topology():

    "Create a network."
    net = Mininet(controller=RemoteController, link=TCLink, accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8',position='10,20,0',range='10')
    sta2 = net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8',position='50,20,0',range='10')
    sta3 = net.addStation('sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/8',position='30,20,0',range='10')
    ap1 = net.addAccessPoint('ap1', ssid='ssid-ap1', mode='g', channel='1', position='15,30,0',range='20')
    ap2 = net.addAccessPoint('ap2', ssid='ssid-ap2', mode='g', channel='6', position='55,30,0',range='20')
    c1 = net.addController('c1', controller=RemoteController,ip="192.168.6.3",port=6633)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()


    print "*** Creating links"
    net.addLink(ap1, ap2)
    net.addLink(ap1, sta1)
    net.addLink(ap1, sta2)
    net.addLink(ap2,sta3)
    print "*** Starting network"
    net.build()
    c1.start()
    ap1.start([c1])
    ap2.start([c1])
```

```
    net.plotGraph(max_x=160,max_y=160)

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



    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Fig.3.15 Código Python de topología de red del tercer escenario de pruebas. Fuente: Autor.

En este ejemplo se crean dos estaciones y tres APs, a cada nodo se le añade la información de posición y rango de alcance para mostrar su ubicación espacial en la gráfica de Mininet-WiFi. Las estaciones sta1 y sta2 se asocian al ap1 y la estación sta3 se asocia al ap3 y entre ambos puntos de acceso se crea un enlace alambrado. Aquí se ejemplificará una función interesante que brinda el Mininet-WiFi, la cual consiste en control de asociación "*associationControl*", un mecanismo que optimiza el uso de los AP, esta asociación puede ser "*ssf*" (*Strongest Signal First)* donde la estación se va a asociar al AP del cual perciba la señal más fuerte o "*lsf*" (*Least Loaded First*) donde la estación se asociara al AP que tenga menos enlaces y en este caso se podrá notar como esta función afectara la topología creada.

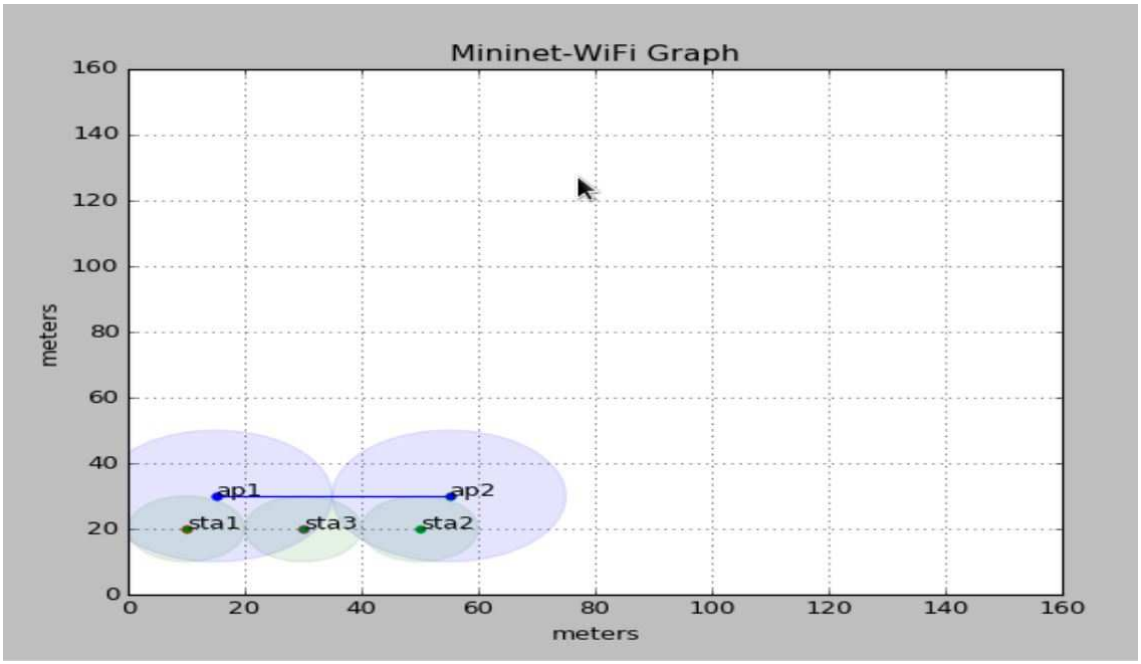

Fig.3.16 Visualización de la topología del tercer escenario de pruebas. Fuente: Autor.

En la gráfica de la figura 3.16 se distingue la ubicación espacial de cada nodo, se aprecia como sta1 y sta3 se encuentran conectadas a ap1 mientras que sta2 está conectada a ap2, discrepando con el código de la topología, esto ocurre porque las estaciones quedan estrictamente en el área de cobertura de los APs a los cuales se encuentran conectados.

Ahora se modificará el código de la topología creada para incrementar el rango de alcance de los APs a 40 metros y se añadirá otra estación (sta4):

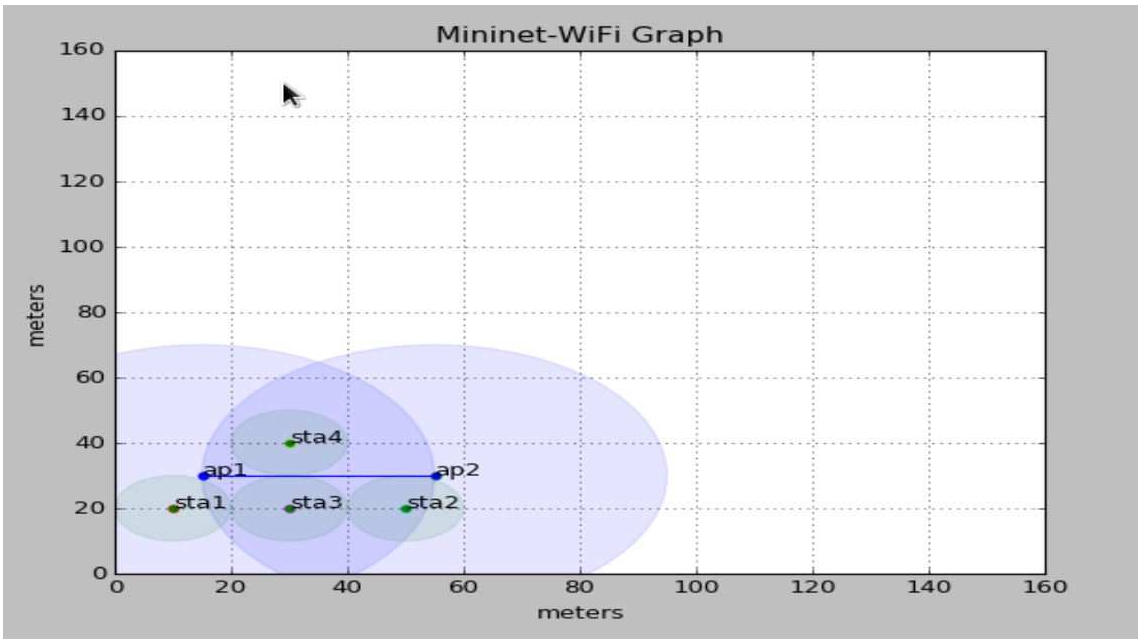

Fig.3.17 Visualización de topología modificada de tercer escenario de red. Fuente: Autor.

En este caso como se ve las estaciones sta3, sta4 y sta2 se encuentran en la convergencia de las áreas de cobertura de ambos APs.

Para comprobar a que AP se encuentra conectada cada estación se empleará el comando *sta iw dev sta-wlan0 link :*

```
mininet-wifi> sta2 iw dev sta2-wlan0 link
Connected to 02:00:00:00:05:00 (on sta2-wlan0)
        SSID: ssid-ap2
        freq: 2437
        RX: 42744 bytes (696 packets)
        TX: 989 bytes (11 packets)
        signal: -30 dBm
        tx bitrate: 18.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi> sta3 iw dev sta3-wlan0 link
Connected to 02:00:00:00:04:00 (on sta3-wlan0)
        SSID: ssid-ap1
        freq: 2412
        RX: 54906 bytes (938 packets)
        TX: 989 bytes (11 packets)
        signal: -30 dBm
        tx bitrate: 18.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi> sta4 iw dev sta4-wlan0 link
Connected to 02:00:00:00:04:00 (on sta4-wlan0)
        SSID: ssid-ap1
        freq: 2412
        RX: 74367 bytes (1250 packets)
        TX: 989 bytes (11 packets)
        signal: -30 dBm
        tx bitrate: 18.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

Fig.3.18 Verificación de la conexión de las estaciones del tercer escenario de pruebas. Fuente: Autor.

Es evidente que cada estación se encuentra conectada al punto de acceso del cual percibe la señal más fuerte. Si se quisiera equilibrar la carga de los AP una opción factible seria desconectar sta4 de ap1 y conectarlo a ap2 como se muestra en la siguiente figura:

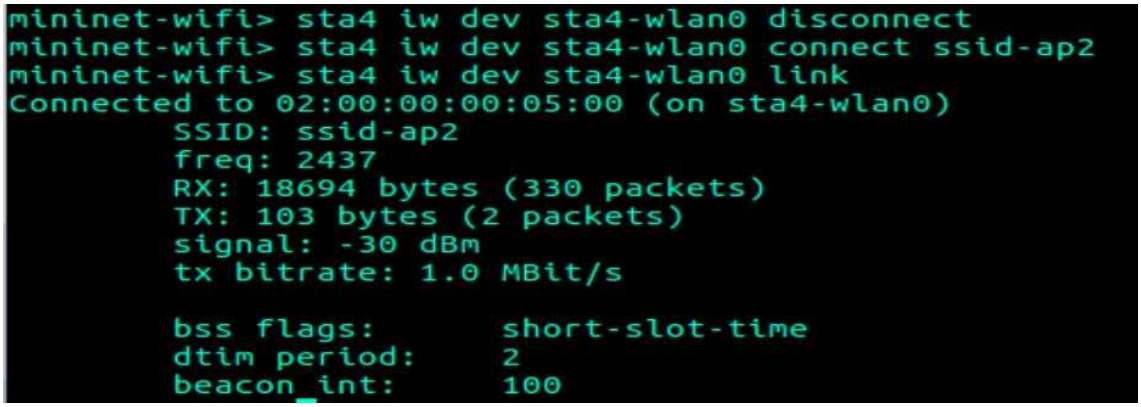

Fig.3.19 Reconexión de una estación a otro Punto de Acceso. Fuente: Autor.

Por último, para ver cómo responde el sistema al tráfico se realizará un ping desde sta2 a sta4.

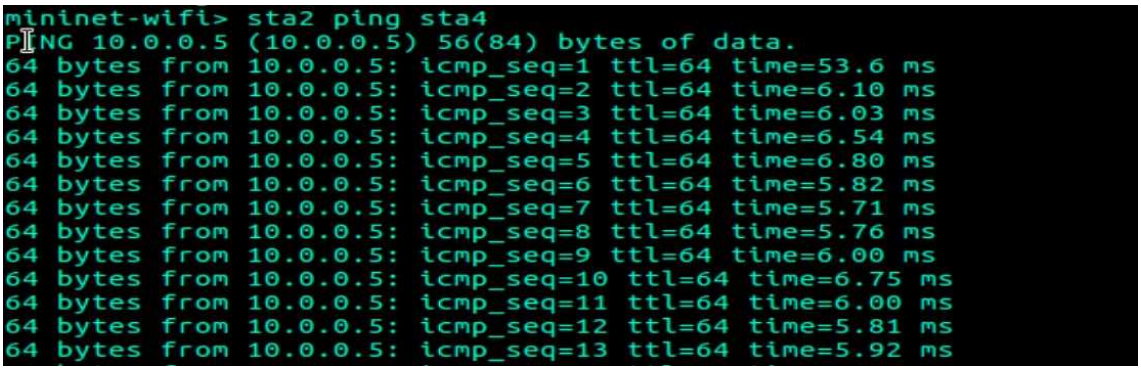

Fig.3.20 Prueba de ping entre dos estaciones. Fuente: Autor.

El *Ping* llega a su destino, luego se iniciará un servidor *iperf* en la estación sta2.

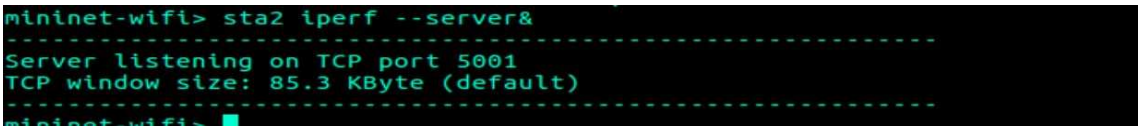

Fig.3.21 Instalación de un servidor iperf en una estación. Fuente: Autor.

Ahora se ejecutará el comando dump para saber la dirección IP de sta2, donde se inició el servidor *iperf*.

```
mininet-wifi> dump
<Station sta1: sta1-wlan0:10.0.0.2 pid=2713>
<Station sta2: sta2-wlan0:10.0.0.3 pid=2716>
<Station sta3: sta3-wlan0:10.0.0.4 pid=2719>
<Station sta4: sta4-wlan0:10.0.0.5 pid=2722>
<OVSAP ap1: lo:127.0.0.1,ap1-wlan1:None,ap1-eth2:None pid=2728>
<OVSAP ap2: lo:127.0.0.1,ap2-wlan1:None,ap2-eth2:None pid=2731>
<RemoteController c1: 192.168.6.3:6633 pid=2736>
```

Fig.3.22 Verificación de las direcciones IP de las estaciones. Fuente: Autor.

Tras esto se abrirá una ventana de comando en sta4 con el comando *xterm* y se iniciará un cliente *iperf* allí.

```
"Node: sta4"
root@raymond-VirtualBox:~# iperf --client 10.0.0.3  --time 60 --interval 2
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 21] local 10.0.0.5 port 53196 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 21]  0.0- 2.0 sec   640 KBytes  2.62 Mbits/sec
[ 21]  2.0- 4.0 sec   512 KBytes  2.10 Mbits/sec
[ 21]  4.0- 6.0 sec   384 KBytes  1.57 Mbits/sec
[ 21]  6.0- 8.0 sec   640 KBytes  2.62 Mbits/sec
[ 21]  8.0-10.0 sec   640 KBytes  2.62 Mbits/sec
[ 21] 10.0-12.0 sec  1.12 MBytes  4.72 Mbits/sec
[ 21] 12.0-14.0 sec   768 KBytes  3.15 Mbits/sec
[ 21] 14.0-16.0 sec  1.12 MBytes  4.72 Mbits/sec
[ 21] 16.0-18.0 sec  0.00 Bytes  0.00 bits/sec
[ 21] 18.0-20.0 sec  1.25 MBytes  5.24 Mbits/sec
[ 21] 20.0-22.0 sec  0.00 Bytes  0.00 bits/sec
[ 21] 22.0-24.0 sec   896 KBytes  3.67 Mbits/sec
[ 21] 24.0-26.0 sec  0.00 Bytes  0.00 bits/sec
[ 21] 26.0-28.0 sec  1.00 MBytes  4.19 Mbits/sec
[ 21] 28.0-30.0 sec  0.00 Bytes  0.00 bits/sec
```

Fig.3.23 Cliente Iperf en un nodo para comprobar tráfico de datos. Fuente: Autor.

En la figura 3.20 se ve la transferencia de datos en los distintos intervalos cada 2 segundos y el ancho de banda de transferencia en Mbits/s de este cliente al servidor *iperf*.

3.5. Escenario de prueba 4: Movilidad, Control de Asociación y Handover

```
#!/usr/bin/python

'Autor:Ramon Estrada de la Torre'

from mininet.net import Mininet
from mininet.node import Controller, OVSKernelAP,RemoteController
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel

def topology():
    "Create a network."
    net = Mininet(controller=RemoteController, link=TCLink, accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8',position='40,40,0',range=10)
    sta2 = net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8',position='40,60,0',range=10)
    sta3 = net.addStation('sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/8',position='60,40,0',range=10)
    sta4 = net.addStation('sta4', mac='00:00:00:00:00:05', ip='10.0.0.5/8',position='60,60,0',range=10)
    sta5 = net.addStation('sta5', mac='00:00:00:00:00:06', ip='10.0.0.6/8',position='80,40,0',range=10)
    sta6 = net.addStation('sta6', mac='00:00:00:00:00:07', ip='10.0.0.7/8',position='80,60,0',range=10)
    sta7 = net.addStation('sta7', mac='00:00:00:00:00:08', ip='10.0.0.8/8',position='100,40,0',range=10)
    sta8 = net.addStation('sta8', mac='00:00:00:00:00:09', ip='10.0.0.9/8',position='100,60,0',range=10)

    ap1 = net.addAccessPoint('ap1', ssid='ssid-ap1', mode='g', channel='1', position='50,50,0',range=30)
    ap2 = net.addAccessPoint('ap2', ssid='ssid-ap2', mode='g', channel='6', position='70,50,0', range=30)
    ap3 = net.addAccessPoint('ap3', ssid='ssid-ap3', mode='g', channel='11', position='90,50,0',range=30)
    c1 = net.addController('c1', controller=RemoteController,ip="192.168.6.3",port=6633)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    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)

    net.startMobility(time=10, model='RandomWayPoint', max_x=120, max_y=120, min_v=0.3, max_v=0.5)

    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Fig.3.24 Código Python de topología de red del cuarto escenario de pruebas. Fuente: Autor.

En la figura se describe un escenario de movilidad, con 8 estaciones y 3 APs, se crean enlaces entre ap1 y ap2, ap2 y ap3, se emplea la función *associationControl* para las conexiones de las estaciones y el modelo de movilidad usado es el *RandomWayPoint*, este es un tipo de movimiento donde los nodos se mueven alrededor de un punto de referencia como su propio nombre lo indica. En este caso también se ejemplificará el *handover* de las estaciones o la

transferencia desde un Punto de Acceso hacia otro, ya que las estaciones van a estar en continuo movimiento.

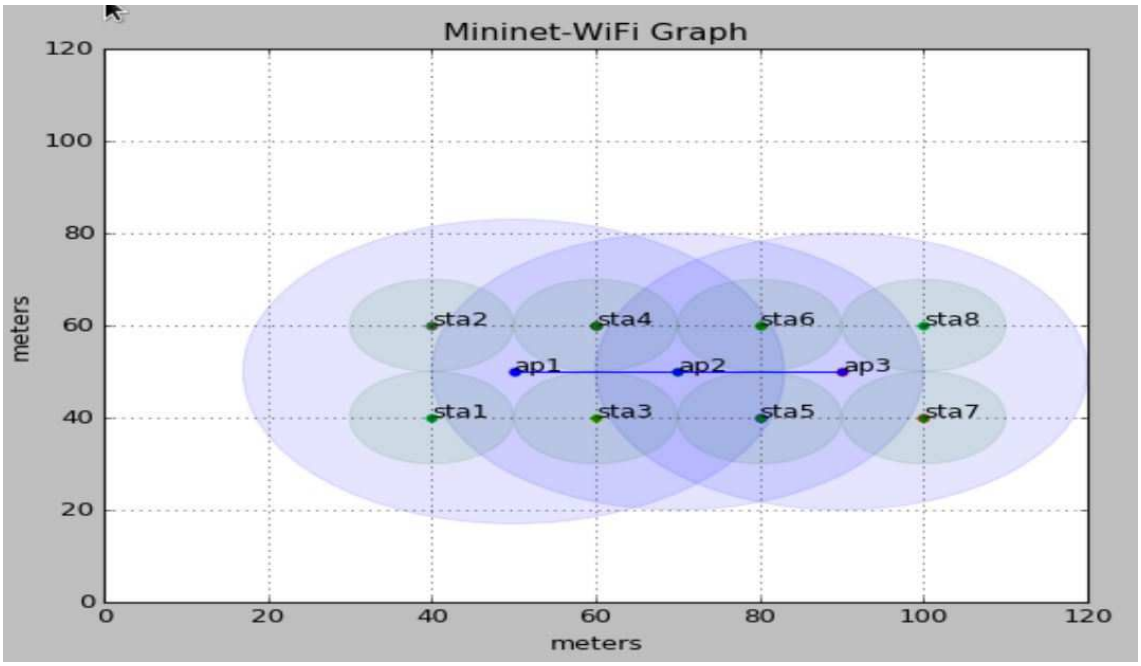

Fig.3.25 Visualización de topología del cuarto escenario de pruebas. Fuente: Autor.

En la figura se ve como en un principio se encuentran ubicados los nodos espacialmente. Las estaciones sta1, sta2, sta3 y sta4 se encuentran conectados a ap1, sta7y sta8 están conectados a ap3, sta5 y sta6 a simple vista no se distinguen a cuál punto de acceso se encuentran conectados, por lo que para saberlo se empleará el comando *iw dev:*

```
mininet-wifi> sta5 iw dev sta5-wlan0 link
Connected to 02:00:00:00:09:00 (on sta5-wlan0)
        SSID: ssid-ap2
        freq: 2437
        RX: 57700 bytes (924 packets)
        TX: 989 bytes (11 packets)
        signal: -30 dBm
        tx bitrate: 48.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi> sta6 iw dev sta6-wlan0 link
Connected to 02:00:00:00:09:00 (on sta6-wlan0)
        SSID: ssid-ap2
        freq: 2437
        RX: 88573 bytes (1458 packets)
        TX: 1067 bytes (12 packets)
        signal: -30 dBm
        tx bitrate: 24.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

Fig.3.26 Verificación de conexión de estaciones del cuarto escenario de pruebas. Fuente: Autor.

Se aprecia que ambas estaciones están conectadas al punto de acceso ap2.

Luego se hará un ping entre sta1 y sta8

```
mininet-wifi> sta1 ping -c1 sta8
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=70.2 ms

--- 10.0.0.9 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 70.264/70.264/70.264/0.000 ms
```

Fig.3.27 Comprobación de conexión entre dos estaciones. Fuente: Autor.

Se observa en la figura anterior que el ping que se envió alcanza su destino.

Tras un tiempo las estaciones cambian de posición y algunas se reconectan a otros puntos de acceso como se ve en la figura, mientras que otras quedan desconectadas al no lograr alcanzar ningún punto de acceso. Ahora se realizará un ping entre las estaciones sta5 y sta7 y otro entre sta6 y sta8.

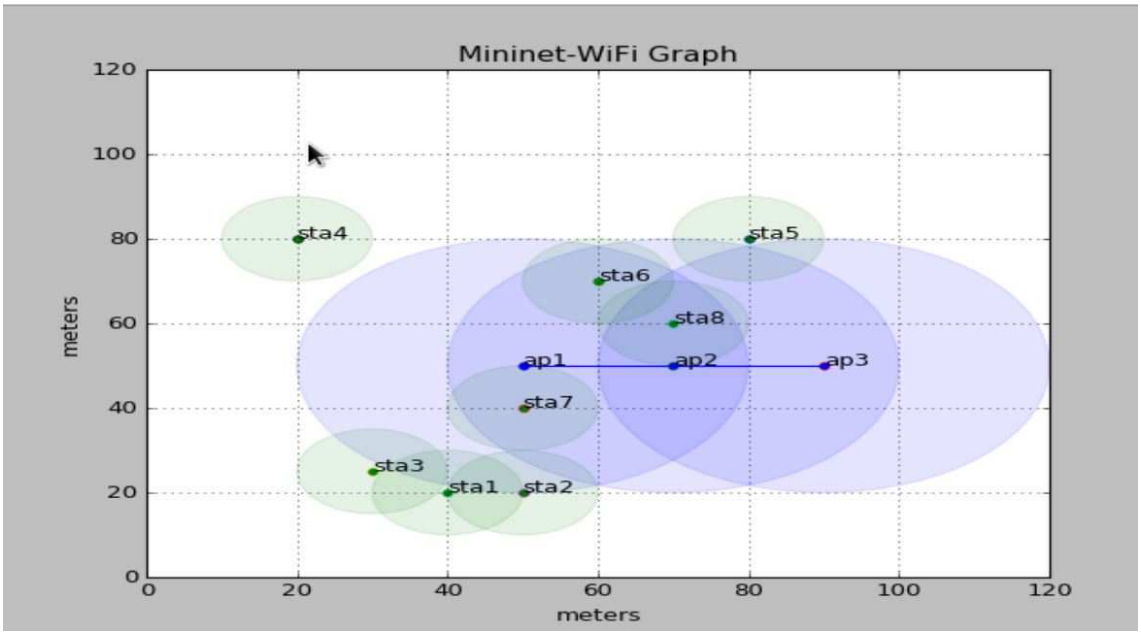

Fig.3.28 Visualización de topología transcurrido un tiempo. Fuente: Autor.

```
mininet-wifi> sta6 ping -c1 sta8
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=31.6 ms

--- 10.0.0.9 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 31.662/31.662/31.662/0.000 ms
mininet-wifi> sta5 ping -c1 sta7
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
From 10.0.0.6 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.8 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

mininet-wifi>
```

Fig.3.29 Verificación de conexión entre dos estaciones. Fuente: Autor.

En este caso el ping entre sta6 y sta8 alcanzó su destino, mientras que el otro no, ya que como bien se puede ver en la figura 3.25 sta5 no se encuentra conectada a ningún AP en ese instante. También se aprecia fácilmente como la estación sta7 se reconecto a ap1 y sta8 a ap2.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

En el presente trabajo se cumplieron con los objetivos propuestos, primeramente se realizó un estudio de las SDN y de las SDWLAN particularmente ,donde se abordaron las principales características y potencialidades de este tipo de redes ,lo cual contribuye a la sistematización de los conocimientos sobre el tema, se diseñaron escenarios virtuales de topologías de redes SDWLAN , luego se simularon en Mininet-WiFi ,comprobando el correcto funcionamiento de la red y del controlador (POX) ,así como todas las funcionalidades que nos brinda esta herramienta .

1. En la ejecución de las pruebas del proyecto se encontraron algunas dificultades importantes para mencionar: para iniciar, la herramienta Mininet-WiFi es muy sensible a la sintaxis a través del lenguaje python, los scripts en python para crear las topologías personalizadas son muy sensibles, cualquier pequeño detalle puede hacer que no corra el emulador; al controlador, en este caso POX, debe asignársele el *datapath* correcto de una interfaz inalámbrica de la máquina virtual, es decir la dirección IP y el puerto por el que estará escuchando, así como activar uno de los módulos que se ajuste a la necesidad de la red de lo contrario también presentará problemas para correr, debido a todo esto se tuvo que invertir mucho tiempo en la validación de las instrucciones escritas llevando a grandes retrasos en la obtención de los resultados.

2. La documentación respecto a SDN en combinación con redes inalámbricas en las diferentes líneas de información es hallada muy genérica, respecto a la meta en este proyecto, la documentación de SDWLAN en particular es muy escasa lo que llevó a

trabajar en el principio prueba y error para ir obteniendo resultados. Es necesario que la comunidad científica siga investigando y desarrollando aplicaciones de SDN en redes WLAN, tema novel del cual queda mucho por innovar y por aportar.

3. Se dieron aportes valiosos para la instalación, ejecución y utilización del emulador Mininet-WiFi y el protocolo OpenFlow 1.0 en conjunto con el controlador POX lo cual puede ser muy útil y servir como apoyo para futuras investigaciones en el tema; al finalizar este proyecto se dispone de una plataforma de pruebas que permitirá a otros el estudio de las SDN y SDWLAN mediante Mininet-WiFi y el controlador POX.
4. SDN permite que gran parte de los dispositivos que conforman las tradicionales redes inalámbricas de área local puedan ser incorporados a una red definida por software con poca inversión económica; se pueden actualizar los firmwares de los dispositivos de la red de datos sustituyéndolos por uno que soporte OpenFlow.

Recomendaciones

- ✓ Se recomienda en futuras investigaciones realizar una comparación de rendimiento y desempeño entre redes WLAN tradicionales y redes SDWLAN empleando recursos de hardware y software reales. Para los routers Linksys de la serie WRT54G ya existe un firmware que trae compilado el protocolo OpenFlow, el *OpenWRT, lo* cual permite implementar escenarios SDWLAN reales con OpenFlow. Este escenario brindaría una oportunidad real para la migración total de la WLAN a una SDWLAN con todas las ventajas que la misma conlleva.
- ✓ Se propone hacer una comparación entre una SDWLAN real y otra SDWLAN emulada, también realizar simulaciones más complejas mediante el Mininet-WiFi utilizando otro controlador más avanzado.
- ✓ Una propuesta interesante seria la utilización de varias interfaces físicas conectadas entre sí junto a escenarios de Mininet-WiFi, permitiendo la comunicación entre ellos.

REFERENCIAS BIBLIOGRÁFICAS

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, pp. 114-119, 2013.

[2] C. J. Bernardos, A. De La Oliva, P. Serrano, A. Banchs, L. M. Contreras, H. Jin, and J. C. Zúñiga, "An architecture for software defined wireless networking," *IEEE wireless communications*, vol. 21, pp. 52-61, 2014.

[3] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 2181-2206, 2014.

[4] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1955-1980, 2014.

[5] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14-76, 2015.

[6] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 493-512, 2014.

[7] (2014). *SCHMITZ, O. Software-Defined Everything: Llevando la inteligencia a un nivel superior*. Available: https://www.linkedin.com/pulse/20140725120736-1573471-software-defined-everything-llevando-la-inteligencia-a-un-nivel-superior

[8] V. ECHEVERRIA and C. DE JESÚS, "DISEÑO Y SIMULACION DE UN PROTOTIPO DE RED DEFINIDA POR SOFTWARE (SDN) USANDO EL PROTOCOLO OPENFLOW," Universidad de Guayaquil Facultad de Ciencias Matemáticas y Físicas Carrera de Ingeniería en Networking y Telecomunicaciones, 2015.

[9] J. C. Chico, D. Mejía, and I. Bernal, "Implementación de un prototipo de una Red Definida por Software (SDN) empleando una solución basada en hardware," *Escuela Politecnica Nacional, Quito, Ecuador*, 2013.

[10] Y. A. Marín Muro, "Plataforma de pruebas para evaluar el desempeño de las redes definidas por software basadas en el protocolo Openflow," Universidad Central" Marta Abreu" de Las Villas. Facultad de Ingeniería Eléctrica. Departamento de Electrónica y Telecomunicaciones, 2016.

[11] M. Abolhasan, J. Lipman, W. Ni, and B. Hagelstein, "Software-defined wireless networking: centralized, distributed, or hybrid?," *IEEE Network*, vol. 29, pp. 32-38, 2015.

[12] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 655-685, 2016.

[13] A. G. Centeno, C. M. R. Vergel, C. A. Calderón, and F. C. C. Bondarenko, "Controladores SDN, elementos para su selección y evaluación," *Revista Telem@ tica*, vol. 13, pp. 10-20, 2014.

[14] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1617-1634, 2014.

[15] "OPENFLOWHUB, “Project Floodlight” " 2011.

[16] *Documentación de NOX*. Available: http://onlab.us/tools_nox.html

[17] *Documentación de NOX y POX*. Available: http://www.noxrepo.org

[18] Documentación de Trema. Available: https://github.com/trema/trema/wiki/Quick-start

[19] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software Defined Wireless Network (SDWN): An evolvable architecture for W-PANs," in *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015 IEEE 1st International Forum on*, 2015, pp. 23-28.

[20] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling sdns," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 1-6.

[21] Y. Li and A. V. Vasilakos, "Editorial: Software-defined and virtualized future wireless networks," *Mobile Networks and Applications*, vol. 20, pp. 1-3, 2015.

[22] C. Niephaus, G. Ghinea, O. G. Aliu, S. Hadzic, and M. Kretschmer, "Sdn in the wireless context-towards full programmability of wireless network elements," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, 2015, pp. 1-6.

[23] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 27-51, 2015.

[24] T. Lei, Z. Lu, X. Wen, X. Zhao, and L. Wang, "SWAN: An SDN based campus WLAN framework," in *Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), 2014 4th International Conference on*, 2014, pp. 1-5.

[25] D. Zhao, M. Zhu, and M. Xu, "SDWLAN: A flexible architecture of enterprise WLAN for client-unaware fast AP handoff," in *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, 2014, pp. 1-6.

[26] K. Akkaya, A. S. Uluagac, and A. Aydeger, "Software defined networking for wireless local networks in Smart Grid," in *Local Computer Networks Conference Workshops (LCN Workshops), 2015 IEEE 40th*, 2015, pp. 826-831.

[27] G. Duarte and R. Lobo, "EMULACIÓN DE ESCENARIOS VIRTUALES, EN UNA SDWLAN (SOFTWARE DEFINED WIRELESS LOCAL AREA NETWORK), DE UN CAMPUS UNIVERSITARIO," *Ingenieria al Dia*, vol. 1, 2015.

[28] D. Zhao, M. Zhu, and M. Xu, "Leveraging SDN and OpenFlow to Mitigate Interference in Enterprise WLAN," *JNW*, vol. 9, pp. 1526-1533, 2014.

[29] D. Zhao, M. Zhu, and M. Xu, "Supporting “One Big AP” illusion in enterprise WLAN: An SDN-based solution," in *Wireless Communications and Signal Processing (WCSP), 2014 Sixth International Conference on*, 2014, pp. 1-6.

[30] M. Copete and J. Emilio, "Estudio del funcionamiento de la herramienta Mininet," 2016.

[31] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," in *Network and Service Management (CNSM), 2015 11th International Conference on*, 2015, pp. 384-389.

[32] R. d. R. Fontes and C. E. Rothenberg, "Mininet-WiFi: A Platform for Hybrid Physical-Virtual Software-Defined Wireless Networking Research," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, 2016, pp. 607-608.

[33] F. R. (2013, Visual Network Description (VND). Available: http://www.ramonfontes.com/visual-network-description/

[34] E. Gregorio Sáinz, "SDN sobre redes IEEE 802.11: simulación mediante MiniNet-Wifi," 2016.

[35] B. Linkletter. (2016, 30/3/2017). *Mininet-WiFi: SDN emulator supports WiFi networks*. Available: www.brianlinkletter.com/mininet-wifi-software-defined-network-emulator-supports-wifi-network

[36] "Andre Panisson. Modelos de movilidad: Mininet-wifi ", 2016.

ANEXOS

Anexo I Instalación de Mininet-WiFi

Para la instalación de Mininet-WiFi sus desarrolladores crearon un script que permite la instalación directa desde el sitio:

```
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 esta última línea de código lo que se está realizando es la elección de los paquetes que se pretenden instalar, los cuales son los siguientes:

W: instala dependencias Mininet-WiFi

n: install Mininet dependencies + core _les

f: instala OpenFlow

3: instala OpenFlow 1.3

v: instala Open Vswitch

p: instala el controlador OpenFlow POX

w: instala Wireshark

Anexo II Instalación Wireshark

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

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

Anexo III Requerimientos e instalación de POX

POX requiere de una versión de Python 2.7. POX corre bajo los Sistemas Operativos Windows, Mac OS y Linux. Muchos desarrollos se han hecho en Mac OS, ocasionalmente se ha probado para los otros sistemas operativos de igual manera; la diferencia existente es el tiempo de como una función se lleva a cabo o de como los reportes de errores se realizan. POX puede ser usado con el intérprete estándar de Python (Python), pero también es soportado en PyPy.

La instalación del controlador POX se realiza a través de las siguientes líneas de comando:

```
sudo apt-get install git
git clone https://github.com/noxrepo/pox
```

Anexo IV Ejecución de POX.

Se deberá ingresar al directorio donde se encuentra POX y a continuación ejecutar el controlador. El *script* que iniciará el controlador es:

```
./pox.py log level –
DEBUG samples.pretty_log forwarding.l2_learning
```

La forma general para inicializar POX es con la siguiente estructura:

```
./pox.py --[opciones] [OpenFlow_versión] --
[IP_controlador — –[puerto] --[otros_componentes]
```

El campo opciones incluye acciones adicionales como activar la funcionalidad de *debug* ,los siguientes componentes indican la version de OpenFlow implementada;la direccion IP del controlador y el puerto por donde se estan escuchando las peticiones de conexión de los dispositivos ; el campo otros_componentes hace referencia a las diferentes funcionalidades que se pueden iniciar ,dependiendo del comportamiento que se requiera en el switch (hub,switch de capa 2,switch de capa 3).En este proyecto se activaran las funcionalidades para que el switch actue como un switch de aprendizaje L2.

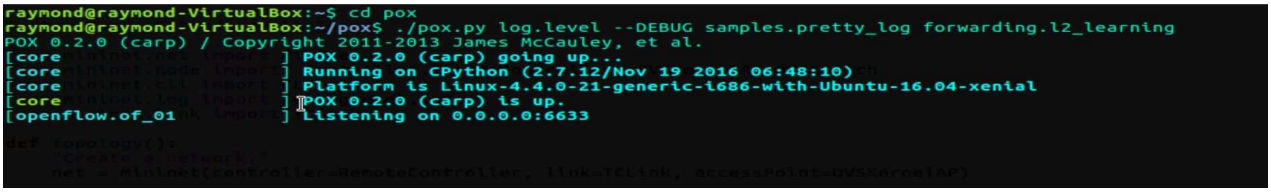

Fig.A1 Ejecución de POX **Fuente: Autor.**

En la figura se observa la inicialización del módulo *forwarding_l2_learning* y que se estableció la conexión con el switch virtual.

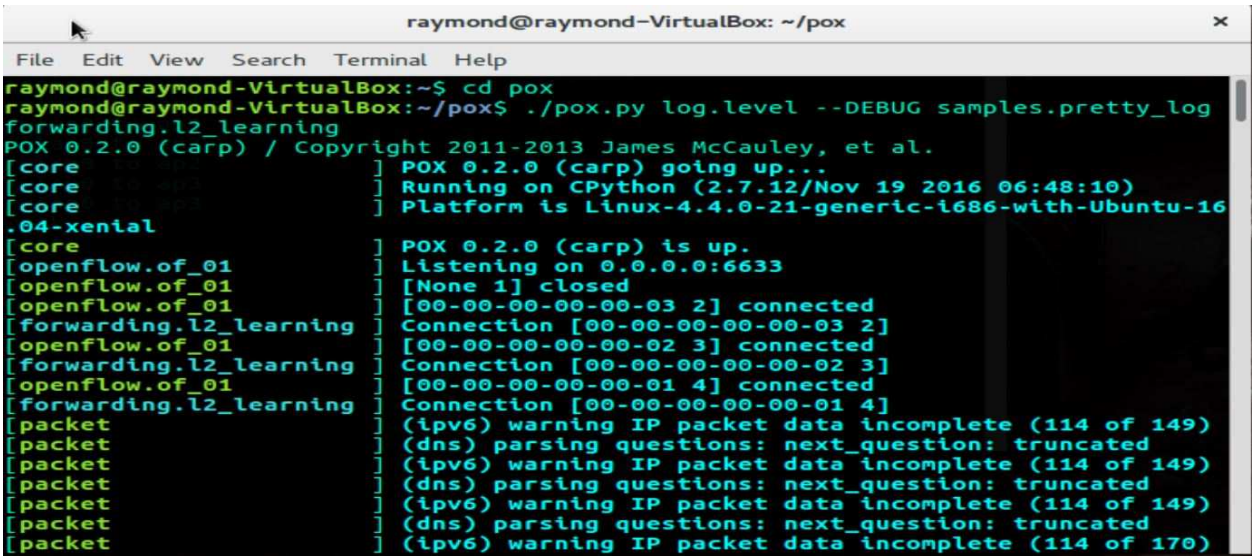

Fig.A2 Mensajes generados por POX **Fuente: Autor.**

En esta figura se ven los mensajes que genera POX. En la línea 1 se muestra la versión, el año y el autor. En 2 ,3 y 4 se inicia POX sobre Ubuntu 16.04. En 5 y 6 el controlador arranca y comienza a escuchar en el puerto 6633. De 7 a 13 se establece la conexión con los Puntos de Acceso y en las líneas restantes se realiza la instalación de los flujos relacionados con el módulo *forwarding. l2_learning*.