UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN
REDES Y SEGURIDAD EN COMPUTO

USO DE SDN, ALGORITMOS DE CLUSTERING Y COLORACIÓN DE GRAFOS PARA LA ASIGNACIÓN DE CANALES WIFI

TESIS

QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
ALEJANDRO ZAMBRANO ESPINOSA

DIRECTOR DE TESIS
Dr. Luis Francisco García Jiménez

Ciudad Universitaria, CD.MX. Enero 2023

Agradecimientos

A mi familia, gracias por su apoyo y soporte a lo largo de mi educación universitaria por siempre impulsarme a seguir adelante.

Agradezco también al Dr. Luis Francisco García Jiménez por su consejo y ayuda como asesor de esta tesis, así como su apoyo brindado a lo largo de la carrera y el posgrado.

Agradezco al CONACYT por el apoyo económico indispensable brindado durante todo el transcurso del posgrado, así como al proyecto DGAPA-PAPIIT IA102822.

Índice general

Índice de figuras

Índice de tablas

Resumen

Con el constante crecimiento en el número de dispositivos que hoy en día pueden conectarse a la red como computadoras, tabletas, teléfonos, impresoras, entre muchos otros, la demanda del espectro dentro de las bandas de frecuencia de WiFi 2.4 y $5.7\,\mathrm{GHz}$ ha aumentado de manera considerable, lo que provoca interferencia entre los AP que brindan el servicio de conexión a Internet. Esto conlleva a una competencia por el ancho de banda que termina en una cantidad considerable de paquetes desechados. Por si fuera poco, WiFi se encuentra dentro de la banda ISM, lo que la obliga a compartir el medio con otras tecnologías como los hornos de microondas y *bluetooth*.

Por otro lado, la tecnología de las Redes Definidas por Software (SDN) es un nuevo paradigma que permite un enfoque novedoso para el control y gestión centralizado de redes. Hoy en día podemos ver presente esta tecnología en los servicios de almacenamiento en la nube. Ejemplos de estas plataformas están Azure o AWS que ofrecen infraestructura como servicio (IaaS). Gracias al principio de virtualización en esta tecnología, se puede controlar y mantener redes de gran tamaño, además de escalar las mismas de forma sencilla sin tener que tomar en cuenta las licencias y marcas de los equipos que las conforman, ya que las SDN tienen la virtud de ser multiplataforma. Gracias a esto, se pueden crear redes que se adapten al contexto dinámicamente, donde el cerebro de éstas puede crear conexiones flexibles que cambia de ruta dinámicamente y balancea las cargas de manera inteligente.

En este trabajo de tesis, se utiliza un algoritmo resultado de la combinación de los paradigmas de clasificación por clusters bajo lógica difusa y la teoría de grafos, por medio del cual se realiza la asignación de canales de frecuencia para los AP dentro de una red centralizada, basada en el paradigma SDN. Para la verificación del desempeño del algoritmo planteado, se realiza un seguimiento en el *throughput* de la red y el retardo de paquetes. Estos resultados se obtienen dentro de un ambiente físico construido con ayuda de dispositivos *Raspberry Pi* que funcionan como AP, así como *switches OpenFlow* para la creación de la red.

Tanto las pruebas reales como las simulaciones muestran una mejoría de alrededor de tres veces en cuanto a la métrica de *throughput* una vez aplicado el algoritmo de asignación. Este incremento implica mejores tasas de transmisión y a su vez una disminución en los valores del *jitter*, que se traduce en la reducción de tiempos de retraso en el envío de paquetes.

Abstract

With the exponential growth of the number of devices that today can connect to the Internet, such as computers, tablets, phones, and printers, the spectrum demand within the WiFi frequency bands (2.4 and 5.7 GHz) has increased significantly, causing interference between APs providing the Internet connection service. This situation creates a competition for bandwidth that ends up in a considerable number of discarded packets. This problem emerges since WiFi is within the ISM band, which forces it to share the medium with other technologies such as microwave ovens and *bluetooth*.

Software Defined Network (SDN), on the other hand, is a new paradigm that allows a novel approach to centralized control and management networks. Nowadays, we can see this technology in cloud computing services. Examples of these platforms are Azure and AWS, which can offer infrastructure as a service (IaaS). Thanks to the principle of virtualization in this technology, it is possible to control and maintain large-scale networks without having to pay the cost of licenses. SDNs have the virtue of being cross-platform. Thanks to this characteristic, SDN can adapt to the context dynamically, where the brain of these networks can create flexible connections that change dynamically and balance loads intelligently.

This thesis proposes an algorithm that combines clustering classification under fuzzy logic and graph theory to enhance the allocation of frequency channels in wireless networks based on the SDN paradigm. The performance of the proposed algorithm is computed by using the network throughput and packet delay metrics. These results are obtained within a physical environment with the help of *Raspberry Pi* devices that operate as APs, as well as *OpenFlow switches*.

Experiments and simulations show that the proposed algorithm enhances the network performance. These optimizations are reflected in throughput enhancements as well as improvements in transmission rates of the network while decreasing jitter values, which results in short delays when sending packets.

Capítulo 1

Introducción

Actualmente, el crecimiento de los dispositivos capaces de comunicarse de forma inalámbrica ha traído consigo grandes beneficios en diferentes ámbitos de la vida cotidiana, como son la monitorización remota de sistemas, el control de hogares por medio de sensores inteligentes, o simplemente la conexión a Internet en espacios públicos. La mayoría de estos dispositivos funcionan dentro de las bandas de frecuencia libres o sin licencia, sobre todo las bandas correspondientes a WiFi. Ello implica el despliegue de redes inalámbricas de área local (WLAN) que brindan el servicio de conexión a Internet. Derivado del despliegue masivo de estas redes sin una correcta regulación, es que se presenta el problema de interferencia entre los puntos de acceso (AP). Para contrarrestar este problema existen diferentes métodos tales como control de topología, control de potencia y asignación de canales. En los últimos años han surgido diferentes soluciones para mitigar este problema mediante algoritmos encargados de la asignación de canales. Sin embargo, las arquitecturas tradicionales de WiFi hacen difícil añadir nuevos mecanismos sin modificar el estándar. Como consecuencia de este problema, en este trabajo de tesis se propone un modelo basado en la clasificación de *clusters* y teoría de grafos, bajo un nuevo enfoque de las redes surgido en los últimos años llamado Software Defined Networking (SDN).

La creciente popularidad de los dispositivos con capacidad de $5\,\mathrm{GHz}$ está mitigando este problema en los entornos interiores, donde la penetración a través de las paredes de las señales de alta frecuencia es limitada. Sin embargo, esto no se aplica a los escenarios exteriores ni a las redes en la banda de $2.4\,\mathrm{GHz}$. Las redes WiFi funcionan en las frecuencias (2.4 y $5\,\mathrm{GHz}$) asignadas por un organismo regulador, por ejemplo, la Comisión Federal de Comunicaciones de los Estados Unidos [10]. Cada estandar WiFi (802,11/a/b/g) define un número fijo de canales para su uso en los AP y los usuarios móviles. Por ejemplo, el estandar 802.11 b define un total de 14 canales de frecuencia, de los cuales del 1 al 11 están permitidos en los EE.UU., el resto son canales experimentales para la investigación. Estos canales tienen un ancho de banda de 22 MHz, comenzando por la frecuencia central de 2.412 GHz para el canal 1. Como resultado, varios canales se superponen con frecuencias de canales adyacentes causando interferencia (también conocida como interferencia de canal adyacente). Sin embargo, dentro de estos 11 canales se tienen tres canales ortogonales (canales 1, 6 y 11) que pueden utilizarse simultáneamente sin causar interferencia.

En este trabajo de tesis se utiliza el enfoque de las SDN, lo cual permite una mayor escalabilidad y flexibilidad al momento de implementar una red WiFi. Además de poder gestionar de forma centralizada la red implementada, sin necesidad de trabajar por separado con cada uno de los equipos que conformarán la red. Esto se debe a la apertura que se tiene a la hora de integrar la infraestructura a la red ya que los equipos utilizados no necesariamente tienen que pertenecer a un mismo proveedor o marca. Por esta razón es que se puede tener un control total de la red sin preocuparnos por la compatibilidad de equipos, ya que las comunicaciones de los dispositivos se dejan al protocolo *OpenFlow*, el cual es un estándar.

1.1. Definición del problema

Actualmente, la popularidad de tecnologías como IoT y el constante aumento en el número de dispositivos que pueden conectarse a la red cómo celulares, tablets, computadoras, sensores, vehículos, etcétera, implican una demanda muy grande dentro de las bandas ISM, específicamente la correspondiente al estándar 802.11 (WiFi). Si bien se han creado alternativas como WiFi5 (802.11ac) o WiFi6 (802.11ax) para dar solución tanto a los problemas de repartición del espectro como a las velocidades para un mejor rendimiento, se sigue presentando la saturación del espectro, sobre todo en la banda de frecuencia de los 2.4 GHz, ya que hasta hoy, muchos dispositivos trabajan sobre esta banda, por ende, los AP que proporcionan servicio a estos dispositivos trabajan sobre esta frecuencia. Esto se traduce en una competencia por el uso del espectro, lo cual causa la generación de interferencia entre los AP y que se traduce en un bajo rendimiento de la red para los usuarios. Por ello, el problema de interferencia entre los AP ha sido fuertemente estudiado en la literatura, teniendo como una de sus soluciones la asignación de canales [11].

En esta tesis, se propone la asignación de canales por medio de la combinación de dos técnicas de clasificación o separación, *clustering* y teoría de grafos. Además de la implementación del paradigma de redes definidas por software. Esto con el fin de que el algoritmo sea capaz de asignar de manera eficiente los canales de frecuencia para los AP dentro de una red en tiempo real de manera dinámica.

1.2. Hipótesis

La combinación de un algoritmo de *clustering* y un algoritmo de coloración de grafos para el plano de control de una SDN permite optimizar la asignación de canales en una red WiFi de manera dinámica.

1.3. Metas

1.3.1. Meta general

Optimizar la asignación de canales en redes inalámbricas WiFi a través de la combinación de algoritmos de *clustering* y coloración de grafos basado en el paradigma de redes definidas por software, reduciendo la interferencia entre *Access Points* y maximizando el *throughput* de la red en tiempo real.

1.3.2. Metas particulares

- Implementar un algoritmo para la asignación de canales de WiFi utilizando *clusters* construidos mediante lógica difusa y un algoritmo de coloración de grafos, tomando en cuenta parámetros como potencia de transmisión, umbral de recepción, probabilidad de enlace (*outage probability*) y el canal WiFi actual.
- Realizar la asignación de canales de WiFi por medio del uso de redes definidas por software (SDN) mediante el protocolo *OpenFlow*
- Mejorar el *throughput* de una red inalámbrica por medio de la implementación de una SDN.

1.4. Metodología

Para llevar a cabo de forma satisfactoria los puntos planteados en la sección anterior, se proponen las siguientes etapas del proyecto:

- Realizar la construcción del esquema de mensajería (paquetes *OpenFlow*) que se utilizará para el intercambio de información entre los AP y el controlador (RYU). Por medio de este esquema los AP informan al controlador el estado actual de los parámetros de los dispositivos como son el canal de WiFi, potencia de transmisión, probabilidad de enlace a través del modelo *Outage probability* [12] y umbral de recepción. De igual manera, por medio de estos mensajes, el controlador informa a cada AP el canal asignado una vez que el algoritmo define los nuevos canales para el mejoramiento de la red, dentro de los aspectos de *throughput* y *jitter*.

- Construir el algoritmo para la asignación de canales, el cual consta de dos etapas. En la primera se implementa un algoritmo de *clustering* con lo cual se realiza la separación de AP en pequeños conjuntos. En una segunda etapa, introducir un algoritmo de coloración de grafos para la asignación final de los canales de WiFi en los cuales van a trabajar los AP.

- Creación de la infraestructura con ayuda de Raspberry Pi, para la implementación de un escenario con el cual se realizarán las pruebas para evaluar el algoritmo propuesto. A la par de la construcción de la infraestructura se realizan simulaciones que permitan la optimización de los parámetros que permiten el mejor funcionamiento de los algoritmos de *clustering* y coloración de grafos.

- Evaluar el algoritmo creado con los parámetros previamente ajustados, en el escenario físico creado a partir de las *Raspberry Pi*.

1.5. Contribución

A diferencia de [13], [14], [15], así como en otros artículos recopilados, el desarrollo de las SDN y las soluciones propuestas están implementadas en Minitet [16], el cual, es un emulador de SDN, por lo que no se enfrenta a implementaciones en escenarios reales. Además, Mininet no considera modelos de propagación, por lo que los resultados obtenidos no se ven afectados por los diferentes problemas que se pueden presentar en este tipo de redes como son: la atenuación de la señal, las fluctuaciones lentas, o las multitrayectorias. A diferencia, en este trabajo de tesis se propone un algoritmo para la asignación de canales usando *clusters* mediante el uso de lógica difusa y coloración de grafos en escenarios físicos. Por lo que el algoritmo propuesto se enfrenta a los múltiples problemas en un ambiente real. Para la creación de la SDN se utiliza el hipervisor XEN, el controlador Ryu, la creación de *switches* virtuales multi-capa mediante *Open VSwitch* bajo el protocolo *OpenFlow* montadas en *Raspberries*, las cuales funcionan como AP y *switches* capa 3.

1.6. Descripción del contenido

- El capítulo 2 presenta los principios y conceptos básicos para entender el paradigma de las Redes Definidas por Software (SDN) y el protocolo OpenFlow. Además de presentar una revisión de algunos de los trabajos relacionados a la asignación de canales empleando el paradigma SDN.

- En el capítulo 3, se describen los principales algoritmos usados para la creación del protocolo de asignación de canales, así como la construcción del mismo. También se describe el proceso de construcción de los mensajes de comunicación entre el controlador y los AP para el intercambio de información y funcionamiento del protocolo de asignación de canales.

- El capítulo 4 presenta la construcción de los escenarios para la implementación de las pruebas del protocolo de asignación. Se muestran los resultados obtenidos en cada una de las pruebas realizadas y se analiza el rendimiento obtenido en cada una de estas para determinar el éxito del protocolo construido.

- Finalmente, en el capítulo 5, se presentas las conclusiones generales y la verificación de la hipótesis.

Capítulo 2

Antecedentes

En este capítulo, se definen los conceptos más destacados del paradigma de Redes Definidas por Software (SDN), así como la arquitectura SDN aplicada a las redes inalámbricas. Se presenta una breve descripción sobre el estándar WiFi 802.11 y sus características. Además, se analizan en profundidad las características principales del protocolo de comunicación para redes SDN OpenFlow, así como las características principales del controlador elegido para la SDN implementada (Ryu). También se presentan los diferentes componentes de la lógica difusa, así como un ejemplo para mostrar el funcionamiento de este paradigma. Finalmente, se presentan los trabajos más relevantes relacionados con el uso de algoritmos para la asignación eficiente de canales de frecuencia en una red bajo el paradigma SDN.

2.1. Estándar 802.11

Las comunicaciones inalámbricas, debido a sus ventajas y necesidades, siguen siendo hoy en día uno de los campos más ricos en el sector de las telecomunicaciones. Por esta razón, la actividad de investigación y desarrollo en este campo se ha intensificado en las últimas décadas para satisfacer la creciente demanda de conectividad. A continuación se describen brevemente las redes inalámbricas de área local (WLAN) basadas en el estándar IEEE 802.11 [17].

2.1.1. Redes inalámbricas basadas en el estándar IEEE 802.11

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) estableció el estándar IEEE 802.11, el cual es hoy en día uno de los más aceptados para las comunicaciones inalámbricas. Este estándar define la capa física y la subcapa MAC de la capa de enlace de datos del modelo de referencia OSI de la arquitectura de red. En la figura 2.1 se muestra el modelo de referencia detallado de IEEE 802.11, en el cual se presentan las diferentes capas del modelo y el propósito de cada una de ellas.

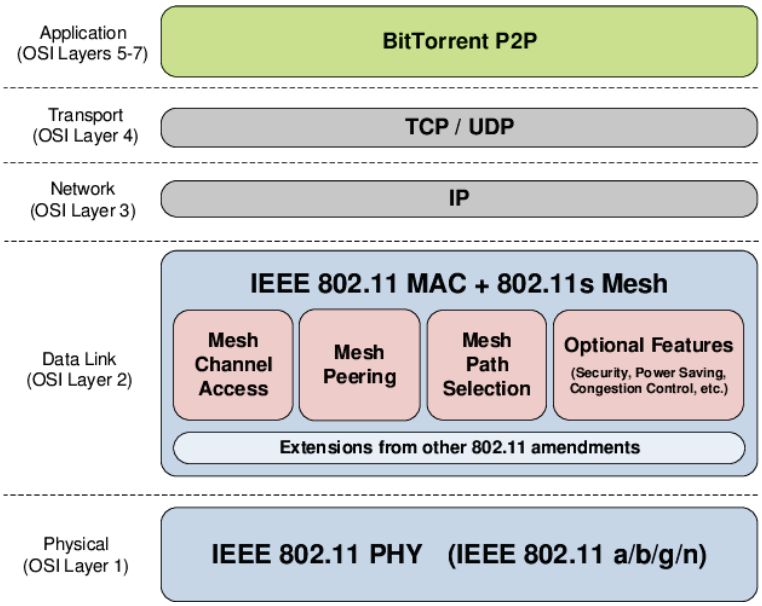


Figura 2.1: Protocolo 802.11 [1].

A lo largo del tiempo han surgido diferentes especificaciones del estándar 802.11, las cuales operan en dos frecuencias principales: 2.4 GHz y 5 GHz. La banda 2.4 GHz no requiere licencia, y está disponible en el mundo entero. Es conocida como ISM (*Industrial*, *Scientific*, *Medical*). Uno de lo problemas con esta banda es que existen muchos dispositivos y/o aplicaciones que operan en ella, provocando interferencia. Por el contrario, la banda de 5 GHz permite canales de frecuencia de mayor ancho de banda y hasta el momento existen menos dispositivos operando en ella. Sin embargo, una de sus desventajas es su área de cobertura más corta comparada con la banda 2.4 GHz, por lo que se necesitan más puntos de acceso para cubrir la misma área. En la tabla 2.1, se muestra una comparación de las características de los principales modos o versiones del estándar IEEE 802.11. Se comparan en términos de frecuencia de operación, ancho de banda de canal y rango de la señal, tanto para escenarios interiores como exteriores. Con respecto a la tasa de transmisión, los datos presentados se refieren a los valores máximos posibles considerando el mejor ancho de banda, esquemas de modulación, esquema de codificación, y número de flujo espaciales (SS, *spatial streams*) basados en *Multiple Input Multiple Output* (MIMO).

	802.11b	802.11a	802.11g	802.11n	802.11ac	802.11ax
Banda (GHz)	2.4	5	2.4	2.4/5	5	2.4/5
Capa Física	DSSS	OFDM	DSSS/OFDM	OFDM	OFDM	OFDMA
Tasa de Transmisión Máxima	Hasta 11 Mbps	Hasta 54 Mbps	Hasta 54 Mbps	Hasta 600 Mbps	Hasta 6930 Mbps	Hasta 9608 Mbps
Número de Streams	N/A	N/A	N/A	4	8	8
Alcance Interior (m)	35	35	38	70	35	30
Alcance Exterior (m)	140	120	140	250	-	120
Ancho de banda (MHz)	20	20	20	20/40	20/40/80/160	20/40/80/160
Año	1999	1999	2003	2009	2013	2018-2019

Tabla 2.1: Comparación de las especificaciones del estándar IEEE 802.11.

Una de las peculiaridades importantes de las redes Wi-Fi es que los canales de frecuencia disponibles están parcialmente solapados. Por ejemplo, en la banda de frecuencia más po-

pular (2.4 GHz), de los 13 canales disponibles (este número puede ser diferente dependiendo de la región), solo un conjunto de ellos no se solapan, estos canales se conocen comúnmente como canales ortogonales. En la figura 2.2 se muestra la distribución de canales del estándar 802.11b, donde los canales coloreados (1, 6 y 11) representan canales ortogonales, lo cual significa que pueden operar sin solaparse uno con otro. Es de importancia destacar que el ancho de banda de cada canal en la banda de 2.4 GHz, es de 22 MHz con un espaciado entre ellos de 5 MHz.

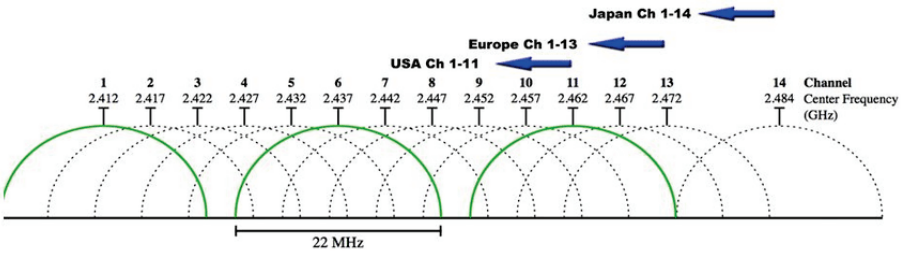


Figura 2.2: Distribución de canales en la banda de 2.4 GHz [2].

2.2. Redes Definidas por Software

2.2.1. Necesidad de redes SDN

Los protocolos de red tradicionales fueron diseñados para adoptar una arquitectura de control distribuido en la que los dispositivos de red deben comunicarse entre sí a través de un conjunto de protocolos de red para negociar el comportamiento en función de la configuración de cada dispositivo individual. Los dispositivos de red son vendidos como componentes cerrados, donde los operadores solo pueden cambiar algunos parámetros de los protocolos de red. Lo que los lleva a traducir las políticas de red de alto nivel en guiones de bajo nivel. Esto se conoce comúnmente como "*Configuration Language*". Además, cada proveedor tiene su propio lenguaje de configuración con su propio cumplimiento del conjunto de normas, lo que da lugar a varios problemas de interoperabilidad. De esta manera, las redes IP tradicionales no son flexibles ni programables por ningún medio. En una red distribuida, de múltiples proveedores, multiprotocolo, y en un entorno dependiente de la configuración manual, la creación de servicios y la resolución de problemas se convirtió en una tarea muy difícil. Más aún, actualmente las redes tienen una integración vertical, esto quiere decir que tanto el plano de control, como el plano de red están integrados dentro de los dispositivos de red. Esto es uno de los principales problemas en las redes, ya que obstaculiza la posible innovación y evolución dentro de las infraestructuras de red.

La tecnología móvil que cada vez se hace más presente, y las nuevas tendencias de virtualización de servidores, obligan a replantearnos los conceptos tradicionales y pensar en nuevas formas de diseño y funcionamiento de las redes actuales. Las Redes Definidas por Software (SDN) son un paradigma de red emergente que da esperanzas de cambiar las limitaciones de la actual infraestructura de red. SDN facilita a los operadores evolucionar las capacidades de la red. Un solo programa de software puede controlar el comportamiento de toda la red, donde esta inteligencia hace posible ofrecer servicios de red como *Networking-as-a-Service* (NaaS), con una reducción significativa de costos de capital y costos operacionales.

2.2.2. SDN

Actualmente, existen varias definiciones sobre las Redes Definidas por Software. Diferentes puntos de vista por parte de organizaciones y empresas líderes del *networking* tienen lugar cuando se trata de definir las SDN. Sin embargo, no es un concepto nuevo ya que varias compañías lo han implementado desde hace varios años, siendo el ejemplo más relevante la

empresa Google [18]. *Open Networking Foundation* (ONF), quien se encarga de la definición del estándar SDN, lo define como la separación física del plano de control, y del plano de datos [3]. En general se puede ver como una arquitectura que se caracteriza por ser dinámica, altamente gestionable, rentable y adaptable en tiempo real, además de poder ofrecer un mayor ancho de banda a las aplicaciones. Una de las características principales de esta arquitectura es la separación de las funciones de control y el reenvío de paquetes en la red, lo que permite una gestión programable donde la infraestructura subyacente pueda ser abstraída de manera fácil para las aplicaciones y los servicios que se encuentren en la red. Más aún, este tipo de redes permiten una gestión centralizada o descentralizada.

2.2.3. Ventajas de SDN

Las SDN tienen muchos beneficios respecto a las redes tradicionales. Una de las principales es la capacidad de gestionar una red desde una perspectiva centralizada. SDN virtualiza tanto el plano de datos como el plano de control permitiendo al usuario disponer de los elementos físicos y virtuales desde un solo lugar. Esto es sumamente útil, ya que la infraestructura tradicional puede ser bastante difícil de monitorizar, especialmente si se tienen muchos sistemas que deben ser administrados individualmente. La inteligencia y las políticas de funcionamiento en las SDN se encuentran centralizadas en un controlador, donde esta entidad puede ser única o estar distribuida. Esto facilita la configuración y administración global de la red debido a que la separación del plano de datos con respecto al plano de control permite la programación y automatización de la infraestructura. Además, existe una notable reducción de gastos de capital y de operación con la adopción de este paradigma puesto que la gestión centralizada simplifica los procesos de administración, configuración y despliegue de nuevas funcionalidades en las redes de datos. También las SDN funcionan muy bien con la virtualización, lo que minimiza aún más la necesidad de adquirir más hardware. Un efecto es que las SDN permiten al usuario escalabilidad. Al tener la capacidad de crear recursos bajo demanda. Esta diferencia es notable cuando se compara con la de una configuración de red tradicional en la que los recursos deben ser adquiridos y configurados manualmente. Las SDN son basadas en estándares abiertos que no dependen del vendedor.

2.2.4. Arquitectura SDN

Las redes definidas por software son arquitecturas de red construidas mediante el uso de software con el fin de controlar la red de forma inteligente y dinámica, ya sea de forma centralizada o descentralizada. La supervisión y administración de las SDN se lleva a cabo mediante un agente conocido como controlador, el cual, administra la red de forma dinámica por medio de la programación de rutinas con las cuales los *switches* que forman parte de la red toman decisiones sobre el flujo de paquetes que se mueven por la red.

La arquitectura de las SDN consisten de 3 planos o capas principales, como se muestra en la figura 2.3. Estos planos se comunican entre sí a través del uso de interfaces conocidas como *Application Programming Interface* (API). De acuerdo al esquema mostrado, la capa superior corresponde al plano de aplicación. La capa de en medio corresponde al plano de control y finalmente, la última capa corresponde al plano de datos o infraestructura. Las comunicaciones entre las capas de control y datos se realizan mediante un conjunto de API conocidas como "*southbound interface*", mientras que la interfaz de comunicación entre el plano de control y el plano de aplicación se conoce como "*northbound interface*".

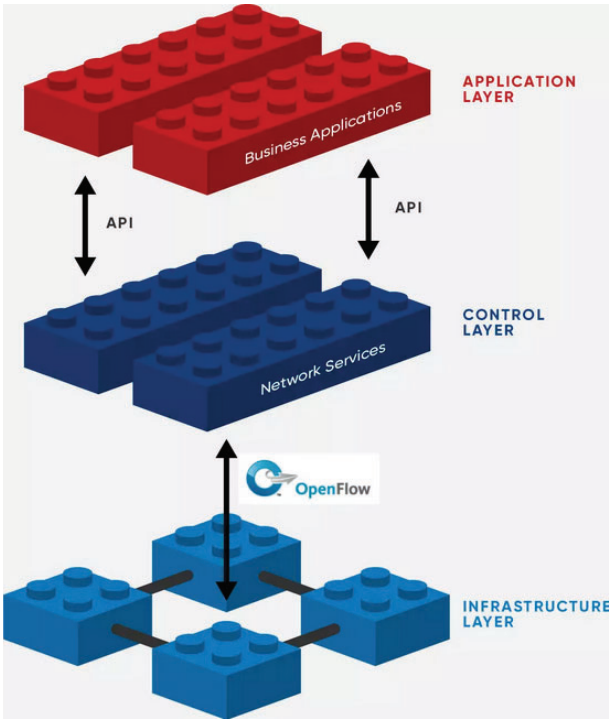


Figura 2.3: Aarquitectura básica de una SDN [3].

Tomando en cuenta que en una SDN el plano de control es independiente de los dispositivos que conforman la red, se puede considerar a este plano como un sistema operativo por medio del cual se pueden realizar distintas tareas como:

- Interoperabilidad de diferentes tipos de dispositivos físicos y virtuales de diferentes proveedores.
- Optimización en la selección de dispositivos, ya sea por precio, o rendimiento, independientemente de las características de los servicios.
- Visibilidad continua de los flujos desde el origen hasta el destino.
- Marco de gestión común para todos los dispositivos.
- Programabilidad para moldear el comportamiento de la red según las necesidades de los usuarios.
- Automatización por políticas.

Plano de aplicación

De acuerdo con la ONF, el plano de aplicación corresponde a programas que comunican al controlador los requerimientos y el comportamiento de la red a través de la *Northbound Interface* (NBI). Además, tiene una vista abstracta de la red con el propósito de tomar las decisiones que se requieran [19]. Estas aplicaciones incluyen tablas de encaminamiento, *firewalls*, balanceadores de carga, monitorización, etc. Esencialmente, una aplicación define las políticas que se traducen en instrucciones específicas que programan el comportamiento de los dispositivos. En este sentido, SDN proporciona una interfaz muy flexible para la creación de nuevos servicios. Para ello, los programadores de red necesitan escribir sus propias políticas y servicios a través de un lenguaje de programación de alto nivel, donde el controlador de red debe ser capaz de traducir estos programas de alto nivel en reglas de reenvío de bajo nivel sobre los dispositivos de reenvío de datos. Ejemplos de estos lenguajes pueden ser C++, Java y Python.

Plano de control

El controlador consiste en una entidad lógica centralizada a cargo de traducir los requerimientos de la capa de aplicación hacia el plano de datos, esta puede incluir estadísticas y eventos. El controlador consta de uno o más agentes NBI, una lógica de control y drivers *Control-Data-Plane Interface* (CDPI) [19]. El plano de control puede ser visto como el cerebro de la red. Toda la lógica de control descansa en las aplicaciones y los controladores, que forman el plano de control. Este elemento realiza las tareas de administración a través de sus diferentes protocolos. Según sea el estado de la red, el controlador se encarga de modificar las reglas para el reenvío de paquetes entre los diferentes dispositivos, con el fin de evitar posibles fallos, balancear las cargas dentro de la red o migrar el trafico de maquinas virtuales. Gracias a estas operaciones, el controlador tiene la capacidad de mantener la red estable con el fin de evitar posibles problemas de seguridad.

El plano de control utiliza un Sistema Operativo de Red (NOS, *Network Operating System*) que facilita la gestión y la administración de red basado en una visión global, el cual instruye a los dispositivos del plano de datos mediante la instalación de entradas de flujo y proporciona información sobre el estado de la red al plano de aplicación para el desarrollo de aplicaciones. El controlador SDN contiene un conjunto de módulos que pueden realizar diferentes tareas que incluyen; administrador de topología, administrador de estadísticas, módulo de encaminamiento, administrador de dispositivos, entre algunas más. Algunos controladores como NOX, POX [20] tienen una arquitectura centralizada y controladores como FloodLight [21] y OpenDaylight [22] son de naturaleza distribuida.

Plano de datos

El plano de datos, es el encargado del reenvío de paquetes y procesamiento de datos por parte de los dispositivos de red lógicos. Está conformado por un agente CDPI y varios motores de reenvió, además de posibles funciones para el procesamiento del tráfico en la red [19].

Debido a la separación del plano de control, las SDN usan en su mayoría *switches* programables. Estos *switches* son capaces de comunicarse con el controlador usando un protocolo llamado *OpenFlow*. Este protocolo consta de dos partes principales: tablas de flujo (los dispositivos pueden tener una o más tablas de flujo), canal seguro (conecta los dispositivos con el controlador). Las tablas de flujo están compuestas por el siguiente formato: coincidencia (*match*), acción, y contadores. Para cada paquete, se busca una coincidencia del encabezado y en base a esta coincidencia se toma una acción particular, y el contador se actualiza en consecuencia. Las instrucciones se instalan en el plano de control utilizando interfaces *southbound*. Bajo estas instrucciones, los dispositivos del plano de datos pueden ejecutar un número de acciones que pueden incluir: reenviar hacia un puerto, reenviar hacia el controlador, descartar el paquete, etc. Tan pronto como el *switch* recibe un paquete, lo primero que hace es comprobar en sus tablas de flujo y ver si su cabecera coincide con una entrada de flujo instalada. Si encuentra dicha entrada, entonces ocurre un *match* que ejecuta la acción automáticamente; de lo contrario, descarta el paquete o lo envía hacia el controlador.

2.3. OpenFlow

Como se mencionó, en la arquitectura SDN, las decisiones de encaminamiento y configuración de la red las toma el controlador desde el plano de control, dejando al plano de datos como función principal el reenvío de paquetes de acuerdo a las instrucciones de sus tablas de flujo. Por tanto, el controlador SDN y los dispositivos que integran el plano de datos (normalmente *switches*), necesitan comunicarse constantemente para el intercambio de mensajes que permiten el correcto funcionamiento de la red. Esta comunicación es posible gracias a una serie de API que operan entre el plano de control y plano de datos. *Openflow* es por mucho, la interfaz SBI (*Southbound Interface*) estandarizada más conocida.

OpenFlow surgió a finales de 2008, y publicó su primera especificación en diciembre de ese mismo año. Es un estándar abierto que ofrece controlar el equipo de red. También ofrece a los administradores de la red, control de los flujos para definir el camino que un flujo lleva desde

el origen hasta el destino. A través del protocolo *OpenFlow* es posible gestionar una red de forma global, ya que es el controlador quien se encarga de comunicar a los *switches* el destino de los paquetes. Con esta tecnología, todas las decisiones referentes al envío de paquetes se encuentran centralizadas, por lo que la red puede ser programada sin importar la marca de los equipos de red.

En un *switch* tradicional, todo el reenvío de paquetes (*datapath*) y el encaminamiento de alto nivel (*controlpath*) se lleva a cabo dentro del mismo dispositivo. Sin embargo, en los *switches OpenFlow* estas dos tareas se separan. Por un lado, el *datapath* reside en el mismo *switch*, mientras que el *controlpath* se crea a través de un controlador (servidor), el cual realiza la toma de decisiones de encaminamiento de alto nivel. Ambas entidades se encuentran en constante comunicación por medio del protocolo *OpenFlow*. Esta metodología de la SDN permite un mejor aprovechamiento de los recursos de la red, en comparación con una red convencional. *OpenFlow* está pensado para trabajar con la movilidad de las maquinas virtuales (VM), redes móviles entre otras.

2.3.1. Arquitectura

Para obtener un control programable sobre el plano de datos, se necesitan *switches* que soporten *OpenFlow* y un controlador que contenga la lógica de la red. En la figura 2.4 se muestra la estructura del protocolo, donde los dispositivos pueden tener una o varias tablas de flujo, un canal seguro (la conexión puede asegurarse empleando protocolos de seguridad de la capa de transporte como SSL) que conecta los *switches* con el (los) controlador(es) y un protocolo que proporciona la comunicación con el(los) controlador(es) externo(s). Un *switch OpenFlow* contiene una o más tablas de flujo y una tabla de grupo, por medio de la cual se realizan las búsquedas de paquetes y el reenvío de los mismos, además de uno o más canales *OpenFlow* para la comunicación con un controlador externo, el cual se encarga de la administración del *switch*. De este modo, el controlador, tiene la capacidad de agregar, borrar y actualizar entradas de flujo. Cada una de las tablas en el *switch* contiene un conjunto de entradas (*flow entries*); cada entrada de flujo esta formada por campos *match*, *counters* y conjuntos de instrucciones (*actions*) para que sean aplicadas a los paquetes.

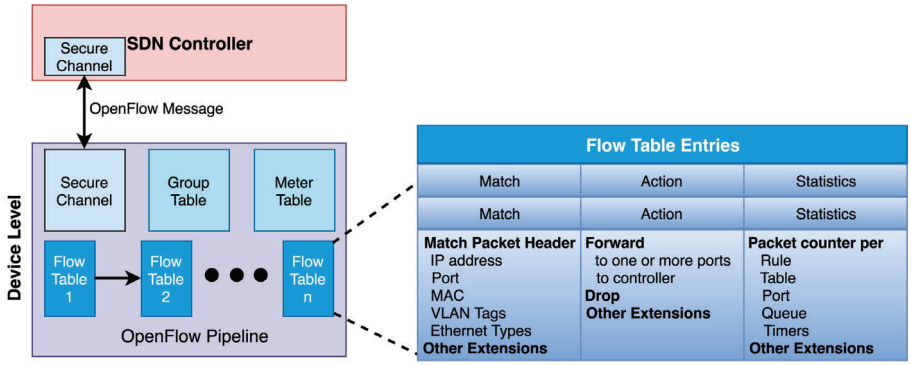


Figura 2.4: Estructura de OpenFlow [4].

Con base en esta coincidencia se toma una acción particular para enviar el paquete a uno o más puertos. Si no se encuentra ninguna coincidencia, entonces se reenvía al controlador usando el mensaje *Packet IN*. Este mensaje contiene la información del puerto de entrada, el encabezado del paquete y el *Buffer ID* donde se almacena el paquete. Para responder al mensaje *Packet IN*, el controlador envía un mensaje *Packet OUT*. Este mensaje contiene el *Buffer ID* del correspondiente mensaje *Packet IN* y las acciones a realizar (por ejemplo, reenviar a un puerto determinado, descartarlo, etc.). Para manejar los paquetes subsiguientes del mismo flujo, el controlador envía un mensaje *Flow MOD* para insertar reglas en la tabla de flujo que le permitan al *switch* saber qué hacer una vez que lleguen ese mismo tipo de paquetes o flujos. Las reglas dadas son comparadas con los paquetes subsiguientes del mismo flujo y se

toma una acción correspondiente. Mientras tanto, el contador se actualiza en consecuencia y se generan estadísticas por regla, por tabla, por puerto, por cola o por temporizador.

2.3.2. Puertos OpenFlow

Los puertos *OpenFlow* son las interfaces de red para intercambiar paquetes en la red. Los *switches* con soporte *OpenFlow* se conectan lógicamente entre sí a través de sus propios puertos, un paquete puede ser reenviado desde un *switch OpenFlow* a otro *switch OpenFlow*. Estos *switches* deben soportar tres tipos de puertos: puertos físicos, puertos lógicos y puertos reservados definido en la especificación 1.5.1 [5].

Los puertos lógicos son abstracciones de nivel que pueden definirse en el *switch* mediante métodos como túneles, o interfaces de *loopback*. Los puertos reservados, por otro lado, especifican acciones de reenvío genéricas como el envío hacia el controlador, la inundación o métodos sin el protocolo *OpenFlow*. Un *switch* no requiere utilizar todos los puertos reservados, algunos son opcionales:

- **ALL** (Requerido)
- **CONTROLLER** (Requerido)
- **TABLE** (Requerido)
- **IMPORT** (Requerido)
- **ANY** (Requerido)
- **UNSET** (Requerido)
- **LOCAL** (Opcional)
- **NORMAL** (Opcional)
- **FLOOD** (Opcional)

2.3.3. Tablas OpenFlow

Dentro de los *switches* compatibles con *OpenFlow* existen dos tipos de tablas:

- **OpenFlow only:** *Switches* que solamente soportan operaciones *OpenFlow*.
- **OpenFlow hybrid:** Estos *switches* admiten el funcionamiento normal de conmutación *Ethernet*, así como el funcionamiento *OpenFlow*.

El *pipeline* de cada *switch OpenFlow* está formado por múltiples tablas de flujos y cada una de ellas contiene múltiples entradas. El proceso de *pipeline* determina cómo los paquetes van a interactuar con las diferentes tablas de flujo (ver figuras 2.5 y 2.6). Un *switch OpenFlow* requiere de al menos una tabla de flujo.

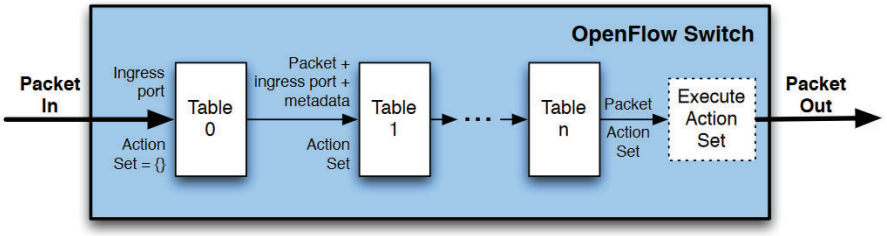


Figura 2.5: Paquetes comparados contra diferentes tablas de flujo en el *pipeline* [5].

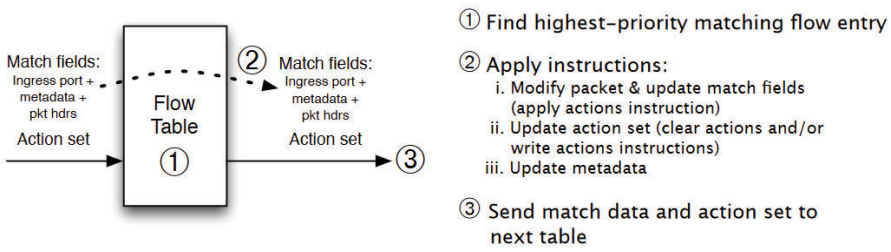


Figura 2.6: Procesamiento del paquete por tabla a través del proceso del *pipeline* [5].

En la versión 1.5 de *OpenFlow* se agregan más componentes al proceso del *pipeline* como se muestra en la figura 2.7. En este caso el proceso de *pipeline* ocurre en dos escenarios: *ingress processing* y *egress processing*. El procesamiento de entrada (*ingress processing*) es el principal proceso que ocurre cuando el paquete entra en el *switch*, y puede implicar una o más tablas de flujo. Mientras el procesamiento de salida (*egress processing*) ocurre después de la selección del puerto de salida, y puede implicar cero o más tablas de flujo.

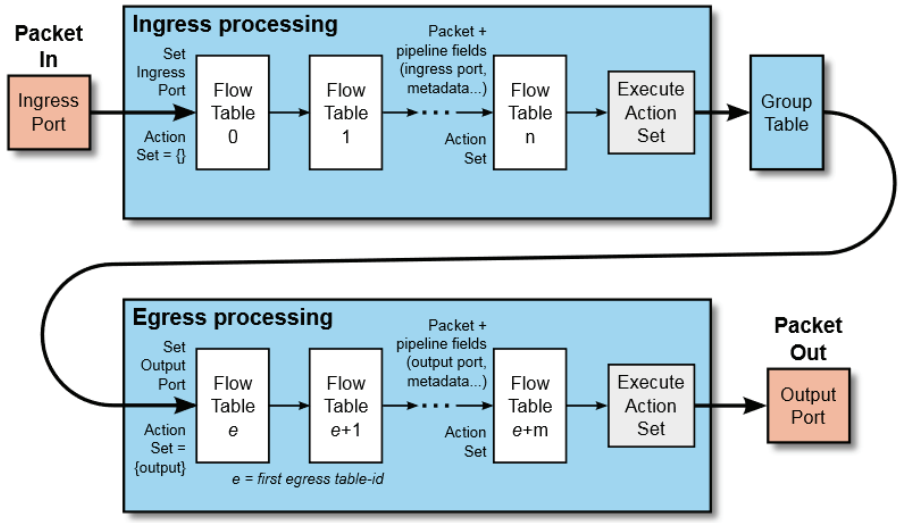


Figura 2.7: Flujo de paquete a través del *pipeline* en *OpenFlow* versión 1.5 [5].

2.3.4. Mensajes OpenFLow

OpenFlow soporta tres tipos de mensajes: *controller-to-switch*, *asynchronous*, y *symmetric*, cada uno con múltiples subtipos.

Controller-to-switch: Los mensajes del controlador al *switch* son iniciados por el controlador y no siempre requieren una respuesta del *switch*. Estos mensajes son usados para la gestión de las tablas de flujo, la configuración del *switch*, así como la petición de información sobre el estado de la tabla de flujo o las capacidades soportadas por el mismo. Algunos de estos mensajes son:

- Features
- Configuration
- Modify-State
- Read-State
- Packet-out

- Barrier
- Role-Request
- Asynchronous-Configuration

Asynchronous: Los mensajes asíncronos se envían sin solicitud del *switch* al controlador y denotan un cambio en el estado del *switch* o de la red, lo que también se denomina como un evento. Uno de los eventos más significativos incluye el evento *Packet IN* que ocurre cuando un paquete que no tiene una entrada de flujo coincidente llega a un *switch*. Al producirse tal evento, se envía un mensaje Packet IN al controlador que contiene el paquete o una fracción del paquete para que pueda ser examinado y se pueda tomar una decisión sobre qué tipo de establecimiento de flujo se puede hacer. Algunos otros eventos incluyen el establecimiento de flujo de entrada y de salida, cambio de estado del puerto u otros eventos de error. Dentro de este tipo de mensajes encuentran los siguientes:

- Packet IN
- Flow Remove
- Port Status
- Role Status
- Controller Status
- Flow Monitor

Symmetric: Finalmente, la tercera categoría de "mensajes simétricos" se envían sin solicitud en dirección, es decir, switch o controlador. Estos mensajes se utilizan para ayudar o diagnosticar problemas en la conexión entre el interruptor y el controlador, y los mensajes de Hello y Echo caen en esta categoría.

- Hello
- Echo
- Error
- Experimenter

Multipart Messages: Debido a que los mensajes *OpenFlow* están limitados a SI64KB, los mensajes *multipart* se utilizan para codificar solicitudes o respuestas que llevan una gran cantidad de datos. Para ello, esto mensajes son divididos en una secuencia de mensajes *multipart*, y se vuelve a reconstruir en el receptor. Estos mensajes se utilizan principalmente para solicitar estadísticas o información de estado del *switch* [5].

Description Message: Este mensaje proporciona información sobre el *switch* como su número de serie, el proveedor, el hardware y un campo de descripción.

La estructura original de este mensaje es la siguiente:

```
/* Body of reply to OFPMP_DESC request. Each entry is a NULL-terminated
* ASCII string. */
struct ofp_desc {
    char mfr_desc[DESC_STR_LEN]; /* Manufacturer description. */
    char hw_desc[DESC_STR_LEN]; /* Hardware description. */
    char sw_desc[DESC_STR_LEN]; /* Software description. */
    char serial_num[SERIAL_NUM_LEN]; /* Serial number. */
    char dp_desc[DESC_STR_LEN]; /* Human readable description of
    datapath. */
};
OFP_ASSERT(sizeof(struct ofp_desc) == 1056)
```

Figura 2.8: Estructura del mensaje de descripción.

Cada parámetro de información del mensaje tienen formato ASCII y esta rellenado con bytes nulos (0) a la derecha. `DESC_STR_LEN` es de 256 bytes y `SERIAL_NUM_LEN` es de 32 bytes.

2.3.5. Establecimiento de la conexión entre el switch y el controlador.

Una vez que un switch está configurado para trabajar con el protocolo OpenFlow, se realiza la búsqueda del controlador enviando un mensaje de sincronización TCP a la dirección IP del controlador en el puerto TCP 6633 o 6653 por defecto. Al recibir el mensaje de confirmación de la sincronización TCP del controlador, el *switch* envía de nuevo el acuse de recibo al controlador, y se produce el *handshake* TCP (figura 2.9). Por lo tanto, cuando cualquier nuevo *switch* se añade a una red *OpenFlow*, se conectará automáticamente al controlador.

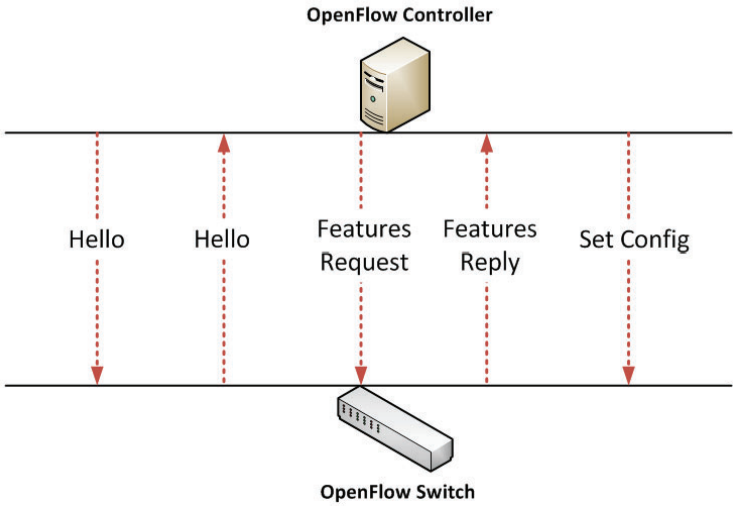


Figura 2.9: Conexión TCP entre el switch y el controlador [6].

Este es el intercambio de mensajes que se realiza para la conexión con el controlador.

- **Hello** (controlador → switch): El controlador envía su número de versión al *switch*.
- **Hello** (switch → controlador): El *switch* responde con su número de versión soportado.
- **Features Request**: El controlador pide ver qué puertos están disponibles.
- **Features Reply**: El *switch* responde con una lista de puertos, velocidad de puerto, y las tablas y acciones soportadas.
- **Set Config**: El controlador solicita al *switch* el plazo del flujo.

2.3.6. Conexión entre hosts en una red OpenFlow

Una vez establecida la conexión entre el *switch* y el controlador, el proceso de comunicación entre 2 o más *hosts* a través de una red *OpenFlow* se lleva a cabo como se muestra en la figura 2.10, donde las flechas representan la dirección de los mensajes.

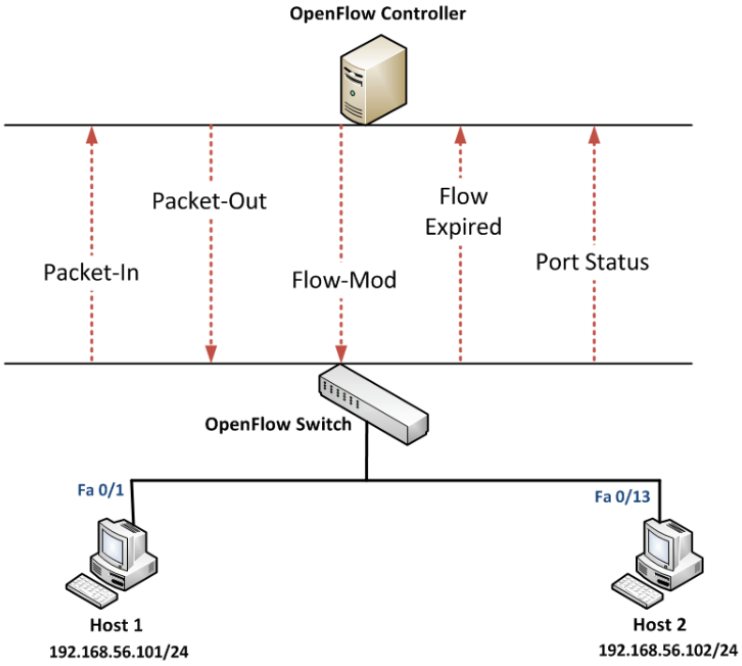


Figura 2.10: Conexión entre *hosts* en una red *OpenFlow* [6].

- **Packet IN**: Cuando cualquier paquete entrante no coincide con ninguna entrada en la tabla de flujo del *switch*, entonces se envía al controlador.
- **Packet OUT**: El controlador envía paquetes a uno o más puertos del *switch*.
- **Flow MOD**: El controlador le indica al *switch* que añada un flujo determinado a su tabla de flujo.
- **Flow Expired**: El *switch* informa al controlador sobre los flujos que han caducado.
- **Port Status**: El switch notifica al controlador sobre la adición, eliminación y modificación de los puertos.

Como se ha mencionado durante este capítulo, *OpenFlow* es un protocolo centrado principalmente en la administración de redes LAN y WAN, con énfasis en los equipos comerciales como *switches*, *routers*, y puntos de acceso. El hecho de que este protocolo sea un estándar flexible y abierto le otorga un papel destacado, ya que permite a sus usuarios tener más libertad para innovar, especialmente en lo que respecta a la academia y la industria. Muchas de estas propuestas están relacionadas con las redes inalámbricas, principalmente porque el protocolo *OpenFlow* no fue diseñado originalmente para las redes inalámbricas. Sin embargo, *OpenFlow* permite conectar múltiples dispositivos de red con la capa inferior del controlador (Ryu en esta tesis) llamada capa de abstracción de servicios (SAL, *Service Abstraction Layer*).

2.4. Ryu

Descripción

Ryu es un *framework* para SDN basado en componentes. Este controlador proporciona diversos componentes de software por medio de una API, la cual permite a los desarrolladores la creación de nuevas aplicaciones de gestión y control para las redes. Ryu tiene la capacidad de soportar varios protocolos para la administración de los dispositivos que pueden formar parte de una red, como *OpenFlow*, *Netconf*, *OF-config*, etc. Acerca de *OpenFlow*, Ryu soporta las versiones 1.0, 1.2, 1.3, 1.4, 1.5 y *Nicira Extensions* [23].

Ryu es un controlador para SDN basado en el lenguaje de programación *Python*. Una de las mayores ventajas de este controlador se encuentra en el lenguaje, al estar desarrollado en *Python* se tiene una gran facilidad para la implementación de múltiples aplicaciones utilizando una gran variedad de bibliotecas que facilitan su creación. Otra característica importante de este controlador es la basta colección de bibliotecas que ofrecen desde la compatibilidad con múltiples protocolos *Sothbound* hasta operaciones de procesamiento de paquetes de red.

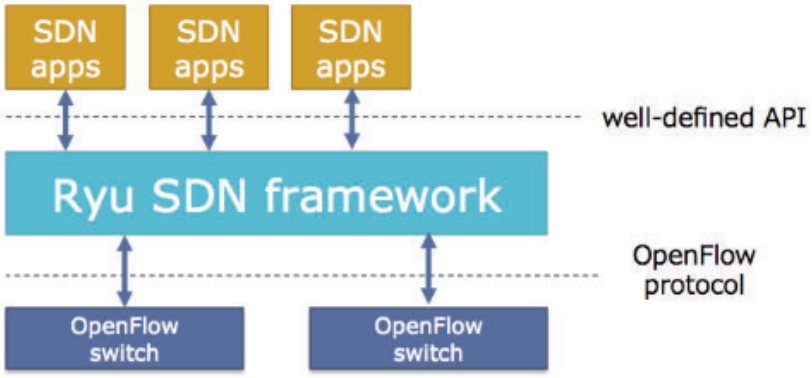


Figura 2.11: Ubicación del controlador Ryu [7].

Arquitectura de Ryu

Con la ayuda de Ryu es posible crear y enviar mensajes *OpenFlow*, monitorizar eventos síncronos o asíncronos, así como poder analizar y manejar el flujo de paquetes de la red. En la figura 2.12 se muestra la arquitectura de este controlador.

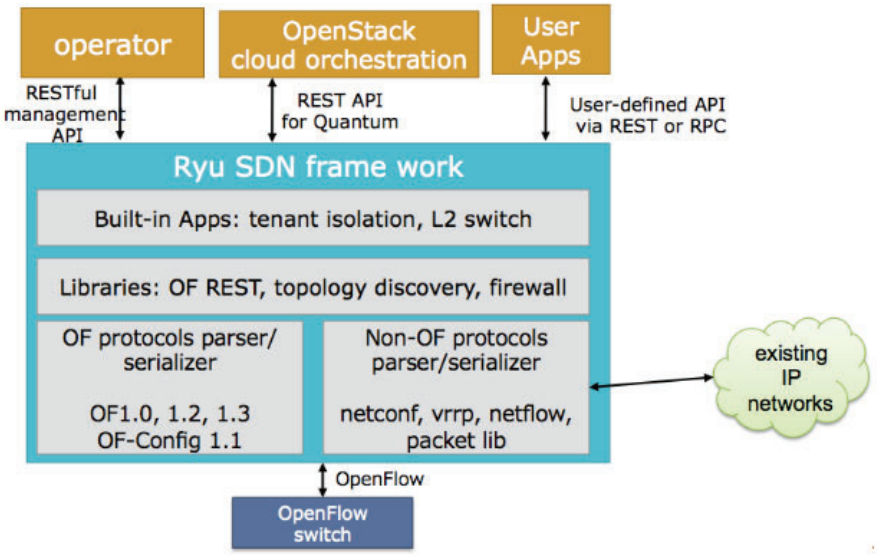


Figura 2.12: Arquitectura del controlador Ryu [7].

Bibliotecas Ryu

Ryu cuenta con una gran variedad de bibliotecas, estas permiten la comunicación entre los diferentes protocolos *SouthBound*. Para el caso de los protocolos *SouthBound*, Ryu admite un conjunto de protocolos como *OF-Config*, *Open vSwitch Database Management Protocol* (OVSDB), *NETCONF*, *XFlow* (*Netflow* y *Sflow*) y otros protocolos de terceros. Admemás la biblioteca Ryu tiene la capacidad de analizar y manejar varios protocolos como *VLAN*, *MPLS*, *GRE*, entre otros.

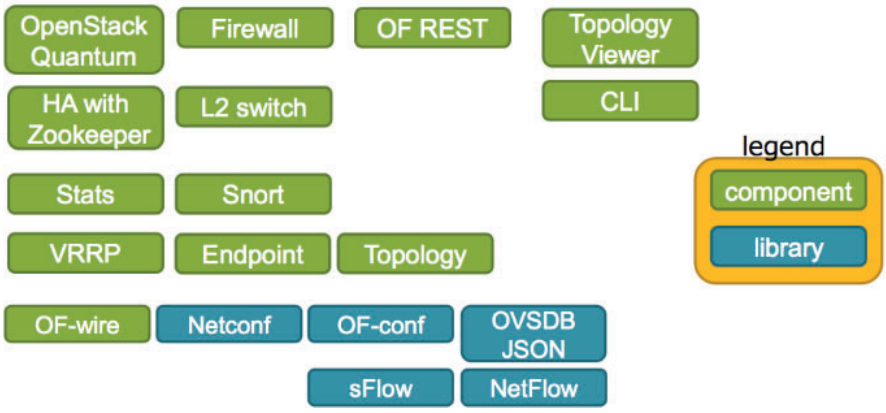


Figura 2.13: Bibliotecas de Ryu [7].

2.5. Algoritmos de coloración de grafos

Un grafo es un par ordenado compuesto por un conjunto de vértices (V) y un conjunto de aristas (E), donde un elemento de V se conecta con otro elemento de V a través de una arista que pertenece a E [24].

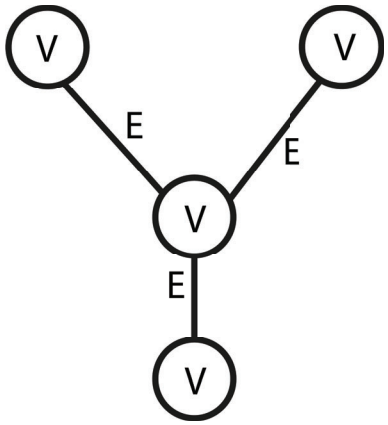


Figura 2.14: Representación de un grafo.

Uno de los problemas más conocidos en la teoría de grafos es el problema de coloración de grafos (*Graph Coloring Problem*). Este problema fue propuesto por Francis Gutrie como el problema de los cuatro colores, el cuál plantea [25] que: cualquier mapa en una superficie plana (o esfera) puede ser coloreado con un máximo de cuatro colores para que no haya dos regiones adyacentes del mismo color.

Al número mínimo de colores usado para colorear un grafo se le denomina número cromático, usualmente representado con la letra λ. La coloración de los grafos se asocia con dos tipos: la coloración de vértices o de aristas. El objetivo en ambos casos es colorear todo el grafo sin conflictos, es decir, que tanto los vértices como las aristas adyacentes no deben tener el mismo color. Los algoritmos de coloración de grafos son utilizados para resolver múltiples aplicaciones de ingeniería, problemas en la ciencia, así como problemas que se presentan en la vida cotidiana. Algunos de estos son, la coloración de mapas, problemas de programación de horarios, problemas de asignación de registros, sudokus y problemas de asignación de frecuencias [26]. Esta última es de interés en este trabajo, puesto que se requiere utilizar un algoritmo de coloración de grafos que asigne eficientemente los canales de frecuencia del estándar 802.11 (WiFi).

2.6. Lógica difusa

La lógica difusa (*Fuzzy Logic*) está basada en la teoría de conjuntos difusos que a su vez es una generalización de la teoría de conjuntos clásicos. La palabra difusa (*fuzzy*) se refiere a parámetros medibles que no son del todo claros. Muchas veces no es posible determinar el estado de un parámetro dentro del rango de verdadero o falso, es decir, en términos de la lógica determinista [8]. Cuando este es el caso, la lógica difusa proporciona mayor flexibilidad para realizar estos cálculos. De esta manera se puede considerar la incertidumbre de cualquier situación.

Otra de las ventajas de la lógica difusa es que la formalización de las reglas se establecen a través del lenguaje natural lo que permite crear razonamientos que pueden tener cierta inexactitud o incertidumbre y aún así brindar un valor final asociado al parámetro medido.

En el sistema de verdad booleano, el 1 representa el valor verdadero, mientras que el 0 representa falso. Sin embargo, en los sistemas difusos se abre la posibilidad de tener un valor que sea parcialmente verdadero o parcialmente falso. Para crear un sistema completo de lógica difusa se requiere de 4 partes: las reglas base, la fuzzificacíon, el sistema de inferencia y la defuzzificación.

Reglas base

Las reglas base se refieren al conjunto de reglas condicionales dadas en el lenguaje natural con las cuales se rige el sistema de toma de decisiones sobre la base de información de

entrada.

Fuzzificación

Este proceso se refiere a la conversión de las entradas, valores nítidos, en valores dentro de los conjuntos difusos. Estas entradas nítidas se refieren a los valores obtenidos o medidas por los sensores o instrumentos de medición ocupados, como temperatura, presión, rpm, voltaje, etc.

Sistema de inferencia

Por medio de este sistema se determina el grado de coincidencia de un valor de entrada difusa con respecto a las reglas establecidas, para después combinar los resultados de las reglas aplicadas y obtener una acción de control.

Defuzzificación

Este proceso regresa un valor nítido o cuantitativo una vez que se aplica el sistema difuso. Los métodos más comunes para obtener este valor nítido o de defuzzificación se listan a continuación.

- IA (Integración adaptativa)
- DBDD (Distribuciones básicas de defuzzificación)
- BDA (Bisectriz de área)
- DDR (Defuzzificación de decisión de restricciones)
- CDA (Centro de área)
- CDG (Centro de gravedad)
- CAE (Centro de área extendido)
- MCE (Método de calidad extendida)
- DDCD (Defuzzificación de clústers difusos)
- MD (Media difusa)
- PM (Primer máximo)
- GLSD (Nivel generalizado de defuzzificación)
- CDGI (Centro de gravedad indexado)
- VI (Valor de influencia)
- UM (Último máximo)
- MDLM (Media de los máximos)
- MDM (Medio de máximo)
- MC (Método de calidad)
- EADM (Elección aleatoria de máximo)
- DSL (Defuzzificación semi-lineal)
- MDP (Media Difusa Ponderada)

De los métodos enlistados previamente, los que se basan en máximos son muy utilizados para el razonamiento difuso. Por otro lado, los métodos de distribución y área tienen la propiedad de continuidad, lo que los hace adecuados para la implementación de controladores difusos.

Función de membresía

Las funciones de membresía realiza el mapeo de los valores nítidos de entrada hacia un valor de pertenencia entre 0 y 1. Los valores que sirven como entrada a la función se conocen como *universo de discurso* o *conjunto universal* (U), este contiene todos los posibles valores de interés para cada aplicación o parámetro particular.

Utilizando estas funciones se determina el grado de pertenencia de un valor de entrada dentro de alguno de los conjuntos difusos. Existen diferentes funciones matemáticas que pueden ser usadas como funciones de pertenencia. Dentro de las más comunes están las siguientes:

- Función Gamma

$$\mu(x) = \begin{cases} 0 & \text{para } x \leq a \\ \frac{x-a}{m-a} & \text{para } a < x < m \\ 1 & \text{para } x \geq m \end{cases}$$

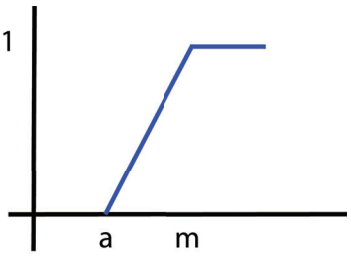


Figura 2.15: Función Gamma.

- Función L

Corresponde a la función en sentido contrario de la función Gamma.

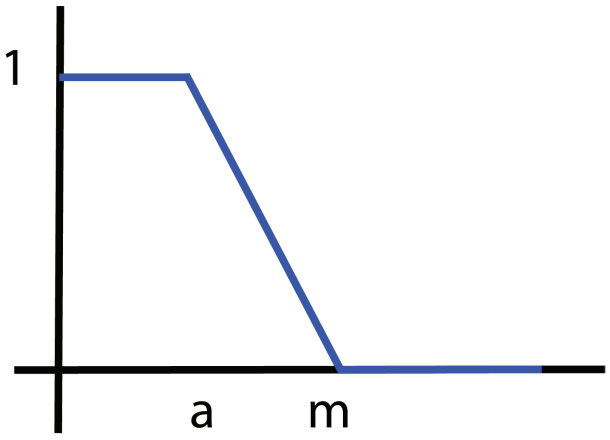


Figura 2.16: Función L.

- Función triangular

$$\mu_(x) = \begin{cases} 0 & \text{para } x \leq a \\ \frac{x-a}{m-a} & \text{para } a < x \leq m \\ \frac{b-x}{b-m} & \text{para } m < x \leq b \\ 0 & \text{para } x > b \end{cases}$$

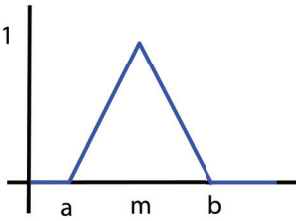


Figura 2.17: Función Triangular.

- Función trapezoidal

$$\mu(x) = \begin{cases} 0 & \text{para } x \leq a \\ \frac{x-a}{b-a} & \text{para } a < x \leq b \\ 1 & \text{para } b < x \leq c \\ \frac{d-x}{d-c} & \text{para } c < x \leq d \\ 0 & \text{para } x > d \end{cases}$$

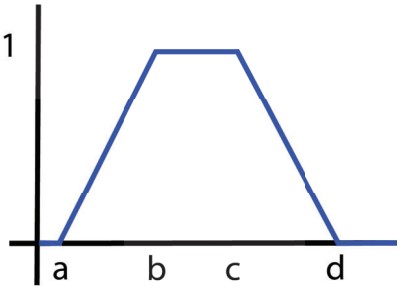


Figura 2.18: Función trapezoidal.

- Función sigmoide

$$\mu_s(x) = \begin{cases} 0 & \text{para } x \leq a \\ 2\left(\frac{x-a}{c-a}\right)^2 & \text{para } a \leq x \leq \frac{a+c}{2} \\ 1 - 2\left(\frac{x-a}{c-a}\right)^2 & \text{para } \frac{a+c}{2} \leq x \leq c \\ 1 & \text{para } x \geq c \end{cases}$$

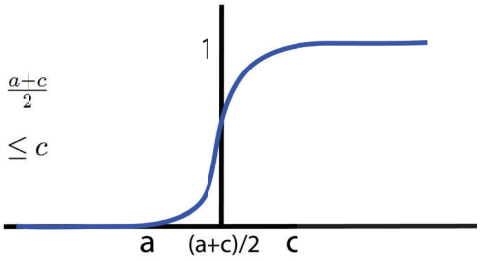


Figura 2.19: Función Sigmoide.

- Función Gaussiana

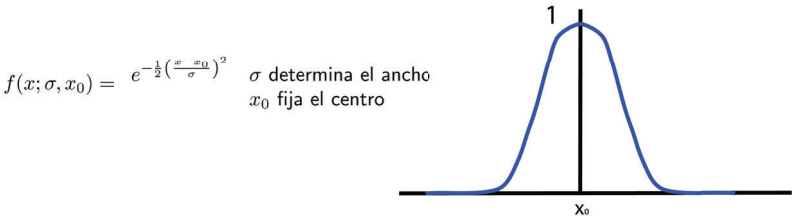


Figura 2.20: Función Gaussiana.

- Función de campana abierta

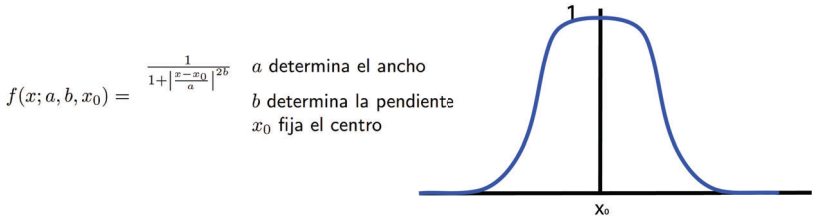


Figura 2.21: Función de campana.

En términos generales, se puede decir que la función Gaussiana corresponde a la versión suave de una función triangular, mientras que la campana generalizada corresponde a la versión suave de una función trapezoidal. Generalmente las funciones triangulares y trapezoidales son usadas cuando el parámetro a modelar permite tener una pendiente discontinua en los cambios de valores de pertenencia. Si el parámetro a modelar no permite cambios repentinos de pendiente, o es necesario calcular la derivada para el mismo, es conveniente usar las funciones suavizadas.

Sistemas de Inferencia Difusa

El sistema de inferencia difusa (FIS) permiten la interpretación del valor de entrada con base en el conjunto de reglas difusas con el fin de asignar los valores correspondientes al vector de salida. Existen 2 tipos de sistemas de inferencia difusa: el sistema de inferencia de Mamdani y el sistema de inferencia de Sugeno.

Sistema de inferencia de Mamdani

Este sistema fue propuesto por Ebhasim Mamdani, donde su principal objetivo fue controlar una máquina de vapor bajo una combinación de calderas a través de varias reglas de control lingüístico generadas a partir de los operadores experimentados. En este sistema, cada regla tiene una salida que pertenece a un conjunto difuso [27].

Sistema de inferencia de Sugeno

Este sistema de inferencia difusa fue propuesto por Takagi, Sugeno y Kang, el objetivo era desarrollar un enfoque que pudiera generar reglas difusas considerando conjuntos de datos de entrada y salida [27]. Como ejemplo de un regla difusa de primer orden de este modelo, se tiene la siguiente:

Si x es A y y es B, entonces $z = f(x, y)$

donde:

- A y B son los conjuntos difusos de los antecedentes.
- $z = f(x, y)$ es una función nítida en el grupo consecuente.

También es posible tener modelos de Sugeno de mayores grados, sin embargo, estos conllevan una mayor complejidad.

Comparación entre los sistemas de Mamdani y Sugeno

Mamdani FIS	Sugeno FIS
Existe una función de membresía de salida	No existe una función de membresía de salida
La superficie de salida es discontinua	La superficie de salida es continua
A través de la defuzzificación de las reglas consecuentes se obtiene un valor específico	No hay defuzzificación. Los valores se obtienen promediando los valores de las reglas consecuentes
Se tiene una interpretación lingüística fácil	Carece de interpretación lingüística
Se puede utilizar para sistemas MISO (Multiple input and single output) y MIMO (Multiple input and multiple output)	Solo funciona para sistemas MISO (Multiple input and single output)
Es adecuado para entradas lingüísticas	Es adecuado para el análisis matemático
Es muy utilizado para el diagnostico médico	Se utiliza para monitorizar el cambio del tráfico aéreo

Tabla 2.2: Comparación de los modelos Mamdani y Sugeno [9].

Proceso de Inferencia Difusa

La tarea de la inferencia difusa es la asignación de una entrada a una salida usando lógica difusa. Este mapeo proporciona una base de conocimiento de la cual se pueden tomar decisiones o discernir patrones. El proceso de inferencia difusa involucra a las funciones de membresía, operaciones lógicas y las reglas Si-Entonces. Uno de los problemas más utilizados para ejemplificar el funcionamiento de los principios de la inferencia difusa es "*el problema de la propina*". A partir de este problema se puede generar un comportamiento complejo a partir de un conjunto compacto e intuitivo de reglas lingüísticas.

El problema consiste en generar un sistema de control difuso que modele la elección del porcentaje de propina que se debe de dejar en un restaurante. Para decidir este valor son considerados 2 parámetros: el servicio y la calidad de la comida, ambos parámetros se califican dentro de la escala de 0 a 10. A partir de estos valores se debe seleccionar un porcentaje para la propina entre el $5\,\%$ y el $25\,\%$.

Por lo tanto, el problema se formula de la siguiente manera [8]:

- **Antecedentes (Entradas)**
  - **Servicio**
    - Universo (rango de valores): Qué tan bueno fue el servicio por parte de los meseros en una escala de 0 a 10.
    - Conjunto Difuso (rango de valores difusos): pobre, aceptable, excelente.
  - **Calidad de la comida**
    - Universo: Qué tan buenos fueron los alimentos en una escala de 0 a 10.
    - Conjunto difuso: mala, decente, deliciosa.
- **Consecuentes (Salidas)**
  - **Propina**

- Universo: Qué porcentaje de propina se debe de dejar en una escala de $5\,\%$ a $25\,\%$.
- Conjunto difuso: baja, media, alta.

- **Reglas**
  - **SI** el *servicio* fue bueno **o** la *calidad de la comida* fue deliciosa, **ENTONCES** la *propina* será alta.
  - **SI** el *servicio* fue aceptable, **ENTONCES** la *propina* será media.
  - **SI** el *servicio* fue pobre **y** la *calidad de la comida* fue pobre, **ENTONCES** la *propina* será baja.

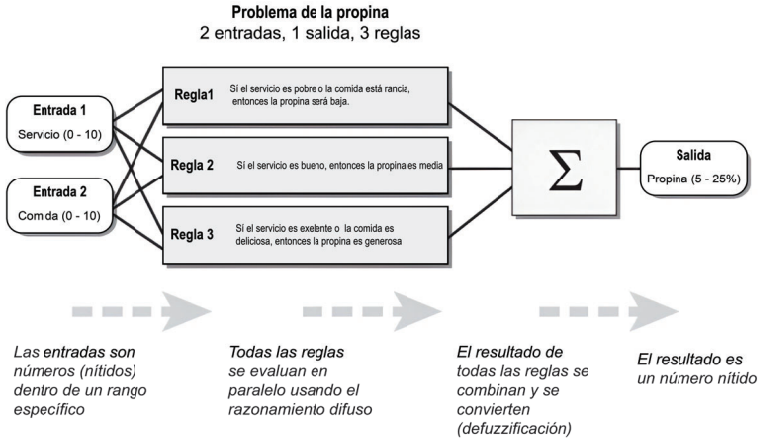


Figura 2.22: Proceso de inferencia difusa [8].

El proceso de inferencia difusa se puede resumir en la siguiente secuencia de pasos:

- Fuzzificación de las variables de entrada.
- Aplicación de los operadores difusos (*AND y OR*) con los antecedentes.
- Implicación de los antecedentes a los consecuentes.
- Integración de los consecuentes a través de las reglas.
- Defuzzificación.

Fuzzificación de las entradas

Este es el primer paso dentro del proceso de inferencia difusa, consiste en tomar valores nítidos como entrada, los cuales serán asignados a valores dentro de los conjuntos difusos, por medio de las funciones de membresía. Si se toma como ejemplo el parámetro de servicio, los valores de entrada se encuentran dentro del rango de 0 a 10. La salida corresponderá a un grado de pertenencia dentro del conjunto difuso (0 a 1). En resumen la fuzzificación de la entrada equivale a una búsqueda de tabla o una evaluación de función.

En el ejemplo propuesto se consideran 3 reglas, las cuales dependen de diferentes conjuntos lingüísticos que caracterizan cada uno de los parámetros utilizados. Por ejemplo, que el servicio sea pobre, que el servicio sea bueno, que la calidad de la comida sea mala, o deliciosa.

Figura 2.23: Fuzzificación de una entrada del parámetro de calidad [8].

En la figura 2.23, se muestra que la calidad de la comida es calificada como deliciosa usando la función de membresía correspondiente. La función de membresía utilizada para el parámetro de calidad corresponde a una función trapezoidal, en la cual se distribuyen los valores de 0 a 10, para determinar la calidad, que puede ir de mala a deliciosa. En este caso, se califica la comida con un 8, que dada la definición gráfica de delicioso a partir de la función de membresía, corresponde a $\mu = 0.7$. De esta manera cada entrada es fuzzificada bajo la función de membresía requerida por las reglas.

Aplicación de los operadores difusos

Una vez fuzzificadas las entradas ya se tiene el valor asociado dentro de los conjuntos difusos que satisfacen cada una de las reglas aplicadas. Si el antecedente de alguna regla tiene varias partes, es decir que el resultado de la fuzzificación de la entrada sean varios valores de pertenecientes a diferentes conjuntos difusos, es necesario aplicar el operador *fuzzy* con el fin de obtener un único valor que represente la regla antecedente. A este número es al que se le aplica la función de salida. Como entrada para el operador *fuzzy* se tienen dos o más valores de pertenencia resultado de la fuzzificación de las entradas. La salida corresponde a un único valor de verdad.

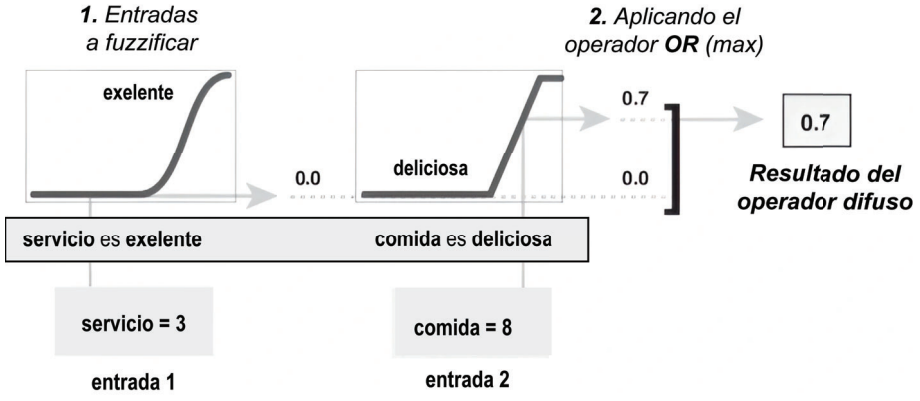


Figura 2.24: Aplicación del operador *OR* [8].

En la figura 2.24, se muestra el uso del operador *OR* (max), evaluando el antecedente de la tercera regla para el cálculo de la propina. Para los parámetros de servicio y calidad de

la comida, los 2 valores antecedentes (servicio excelente y comida deliciosa), producen los valores de 0.0 y 0.7 respectivamente a partir de sus funciones de membresía. Los valores de 3 y 8 para los parámetros de calidad y servicio son los seleccionados por el *comensal*, con los cuales califica el restaurante. Para el caso del parámetro del servicio, se utiliza una función de membresía gaussiana por medio de la cual se pueden obtener las etiquetas de pobre, aceptable o excelente, según el grado de pertenencia. En este caso el operador fuzzy OR selecciona el máximo de los dos valores, 0.7. Por lo cual, después de aplicar el método OR, resulta el valor de 0.7.

Método de implicación

Antes de aplicar este método, se deben determinar los pesos de cada regla (estos van dentro del rango de 0 a 1) , y aplicarlos al número obtenido como antecedente. Por lo general, el peso de las reglas es de 1, esto significa que no existe ningún cambio en el proceso de implicación. Sin embargo, el efecto de una regla relativa al resto puede disminuir si el valor de su peso es distinto de 1.

Después de asignar la ponderación adecuada a cada regla, se implementa el método de implicación. Un consecuente es el conjunto difuso que se representa a través de una función de membresía, con ayuda de la cual se ponderan de manera adecuada las características lingüísticas que se le atribuyen al conjunto. El consecuente es remodelado usando otra función de membresía la cual está asociada con el antecedente (un solo número).

Como entrada al proceso de implicación se tienen un único valor el cual vienen dado por el antecedente, mientras que la salida es un conjunto difuso. Este proceso es implementado para cada regla y admite 2 métodos integrados, que corresponden a las mismas funciones utilizadas en el método AND: **min**(mínimo), el cual trunca el conjunto de salida, y **prod** (producto), que funciona como una escala del conjunto difuso de salida.

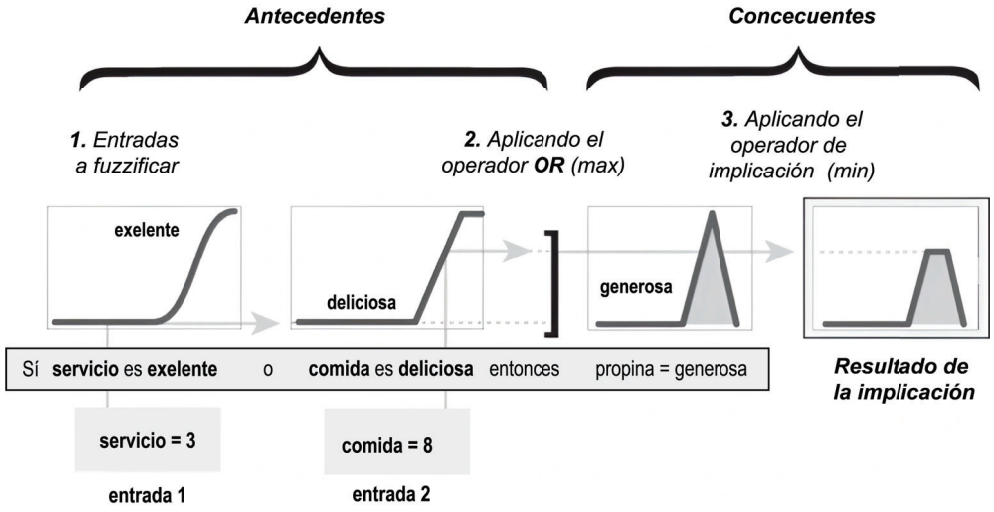


Figura 2.25: Método de Implicación [8].

Integración de las salidas

Ya que las decisiones se basan en probar todas las reglas lingüísticas, los resultados de las reglas se deben de combinar de alguna manera. La integración se refiere a la combinación de los conjuntos difusos resultantes de aplicar cada una de las reglas, estos conjuntos se combinan para obtener uno solo. Este proceso se realiza para cada variable de salida, solo una vez, antes de llegar a la defuzzificación. Por otro lado, como entrada para la integración se utilizan las funciones de membresía devueltas del proceso de implicación.

Ya que el proceso de integración es conmutativo, no importa el orden en que se ejecuten las reglas. Existen 3 métodos diferentes para realizar la integración:

- max (máximo)
- probor (probabilistic OR)
- sum (suma de los conjuntos de salida)

En la figura 2.26, se muestra la agregación de las salidas obtenidas de las 3 reglas para generar un único conjunto difuso cuya función de membresía asigna un porcentaje para los valores de propina.

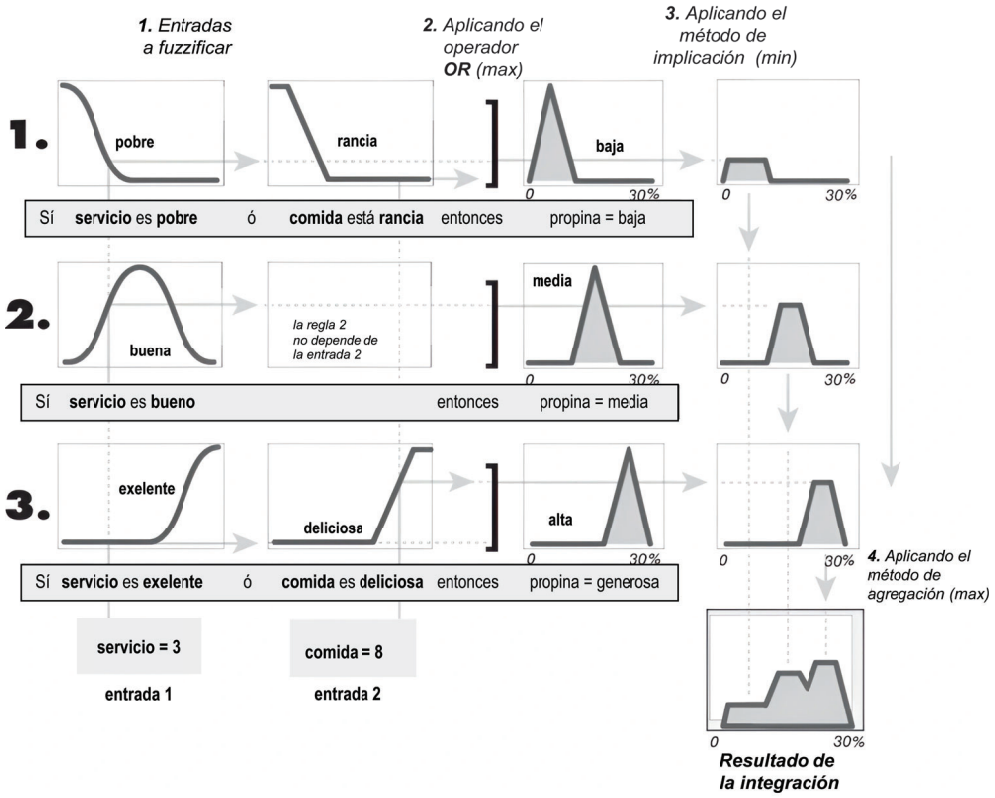


Figura 2.26: Integración de las reglas [8].

Defuzzicación

Para llevar a cabo el proceso de defuzzificación, se utiliza como entrada el conjunto difuso resultante del proceso de integración. Por muy difusos que sean los conjuntos obtenidos en todos los pasos anteriores, al final el resultado para cada variable de salida resultará en un número. Sin embargo, ya que el proceso de integración arroja un conglomerado de valores, estos deben ser defuzzificados para obtener un solo valor.

Existen cinco métodos principales de defuzzificación: centroide, bisector, medio del máximo (el promedio del valor máximo del conjunto de salida), mayor del máximo y menor del máximo. De estos cinco métodos el más utilizado es el centroide, ya que devuelve el centro del área bajo el conjunto difuso, como se muestra en la figura 2.27.

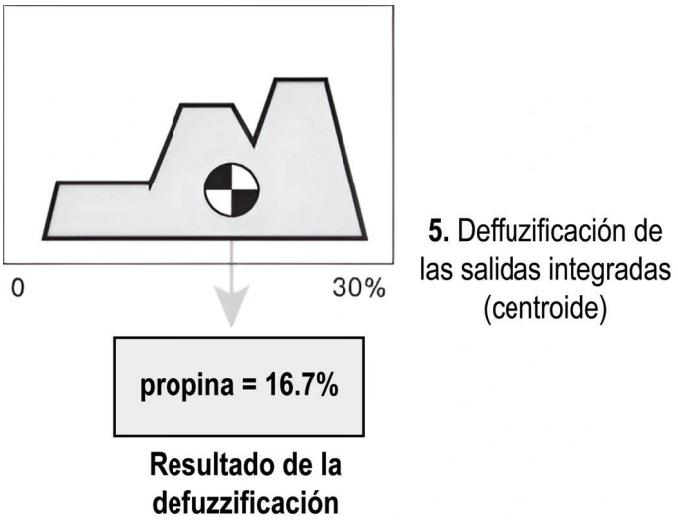


Figura 2.27: Defuzzificación [8].

Diagrama de inferencia difusa

Este diagrama está compuestos por cada una de las partes descritas anteriormente . Aquí se muestran todas las partes del proceso de inferencia difusa.

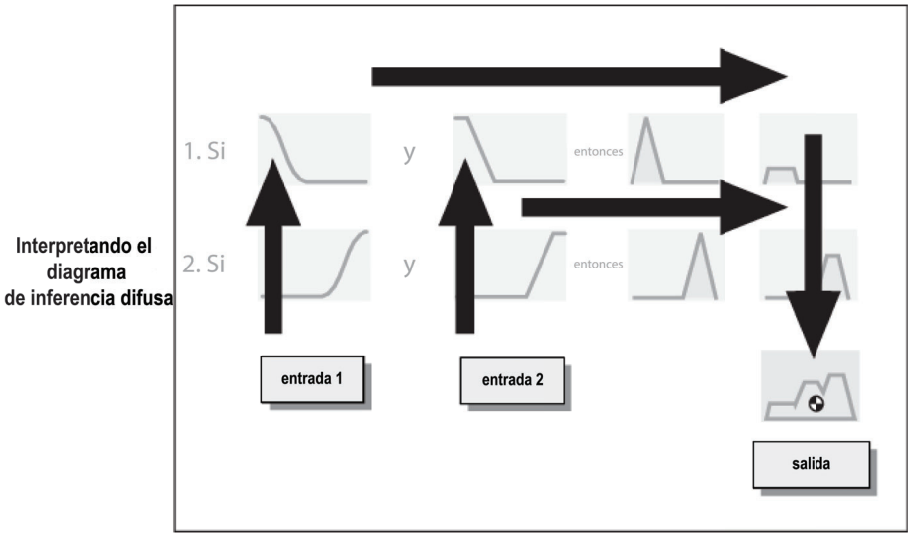


Figura 2.28: Proceso de inferencia difusa [8].

En la figura 2.28, el flujo va de las entradas en la parte inferior izquierda, a través de cada fila, y luego hacia abajo donde se encuentran las salidas de la regla en la parte inferior derecha. Este flujo compacto muestra todo a la vez, desde la fuzzificación de variables lingüísticas hasta la defuzzificación después del proceso de integración.

Finalmente la figura 2.29 muestra de forma completa cada uno de los pasos del proceso de inferencia difusa para *el problema de la propina*.

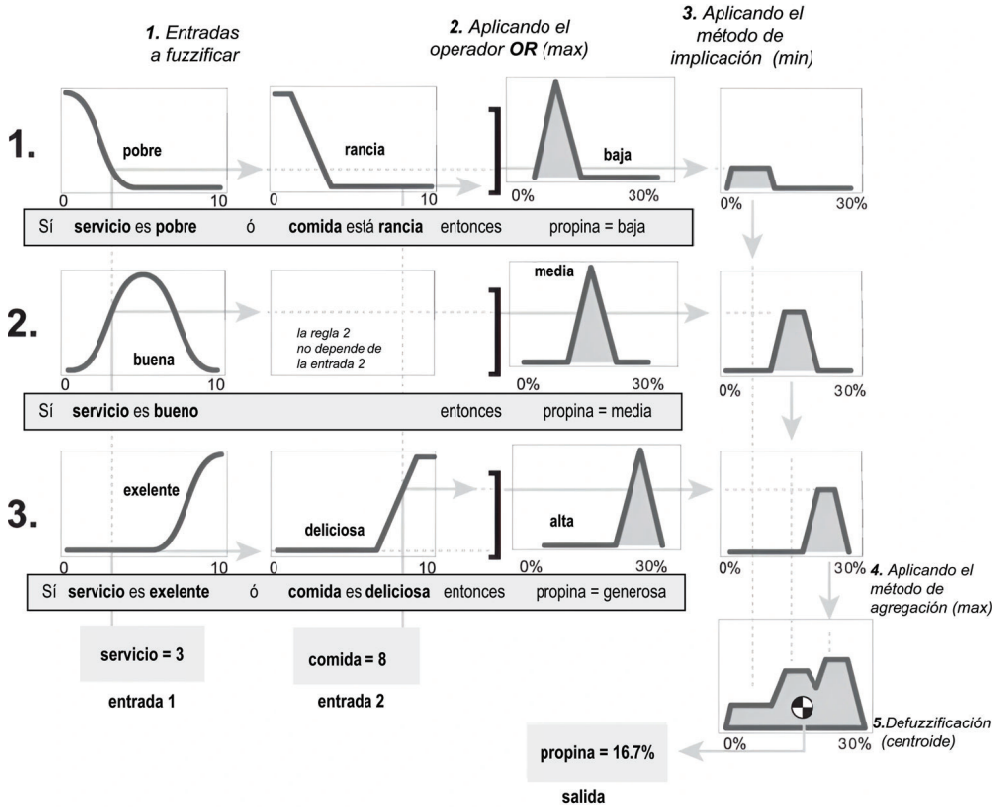


Figura 2.29: Problema de la propina [8].

Del diagrama anterior se puede observar que para las entradas seleccionadas el método de implicación se realiza mediante la función *min*. La función *max* es utilizada para el operador difuso *OR*. Por último, la regla 3 (correspondiente a la tercera fila) tiene una mayor influencia a la salida.

2.7. Trabajos relacionados

En esta sección, se presentan los trabajos relacionados con los temas de SDN en redes inalámbricas y la asignación de canales de frecuencia en redes IEEE 802.11 por medio de algoritmos de teoría de grafos y *clustering*. Dentro del tema de SDN para la asignación de canales con *Machine Learning* (ML), en [28] se muestra la asignación adaptativa de canales y predicción de tráfico en una red SDN-IoT. Este proceso se lleva a cabo por medio del uso de redes neuronales que sirven para clasificar el tráfico de la red, así como realizar la evaluación de los canales para dar una nueva asignación en caso de ser necesario de acuerdo con el tráfico de la red. El uso de algoritmos de ML implica un previo entrenamiento a partir de una base de conocimiento la cual a su vez, tiene que ser filtrada de antemano para asegurar que contenga los datos necesarios para el correcto aprendizaje del algoritmo. Esto significa una gran inversión de tiempo y recursos para la construcción de la base de conocimientos.

En [14], se realiza el proceso de asignación de canales por medio de un algoritmo de *clustering*, el cual se basa en el parámetro de similitud Jaccard. Para ello realiza la comparación de elementos entre dos conjuntos, específicamente parámetros pertenecientes a los *Access Point* como potencia, RSSI, distancia, etc. Sin embargo, estos parámetros no siempre garantizan la formación de *clusters*, ya que, si existe mucha diferencia entre ellos, se formarán *clusters* de un solo elemento. En este trabajo de tesis para garantizar la formación de *clusters* se hace uso de lógica difusa y parámetros de similitud, como se muestra en [29], donde a través de esta herramienta se puede reducir el número de elementos posibles con el fin de mejorar el desempeño de la creación de *clusters*.

En [30], se muestra un algoritmo para la selección de canales en redes $5G$ a partir de la distribución de usuarios secundarios (SU) y de ciertos valores de calidad de servicio (QoS) con el fin de ofrecer a cada SU un cierto QoS en un canal, de tal forma que se tenga baja interferencia.

En [31] se propone la asignación de canales en redes WLAN usando ML. Loa autores propone un algoritmo nombrado *Bad Neighbour Detection* por medio del cual se busca optimizar la asignación de los canales y de esta forma ofrecer a los usuarios una mejor experiencia sobre el desempeño de la red.

Otra de las alternativas para la asignación de canales es el uso de coloración de grafos. Esta técnica permite asignar etiquetas a los nodos de un grafo de tal forma que ningún vértice adyacente comparta el mismo color. En [32] con ayuda de esta técnica y algoritmos de *clustering* se busca asignar los recursos a los usuarios de manera eficiente en redes inalámbricas. A diferencia de los trabajos previos, en esta tesis se utiliza lógica difusa para mejorar la toma de decisión al crear los *cluster* mediante el índice de Jaccard. Esta etapa de clasificación permite identificar de mejor manera los grupos de APque causan más interferencia entre ellos, para que posteriormente se aplique un algoritmo de coloración en cada *cluster* con el fin de minimizar la interferencia entre canales.

Aunque todos los trabajos presentados previamente abarcan el despliegue de SDN en redes inalámbricas, no muestran claramente de que forma se realizaron las implementaciones para la comunicación entre los dispositivos de red, lo cual es un punto importante que se quiere conseguir en este trabajo. Por otro lado, los algoritmos para la asignación de canales presentados en los diferentes trabajos no permite saber si en un escenario real se comporten de manera similar, ya que estos estudios son simulados y no toman en cuenta fenómenos que pueden llegar a generar interferencia en la red.

Capítulo 3

Algoritmos propuestos e Implementación de mensajería

En este capítulo, se presentan los algoritmos utilizados para la asignación de canales dentro de la red. Además, se presenta un análisis de cada uno de ellos con el fin de conjuntar su implementación. Finalmente, se muestra el desarrollo de la implementación de los mensajes de comunicación que serán enviados del controlador Ryu a los puntos de acceso (AP) y viceversa, tanto para la recopilación de información como para la asignación de nuevos canales.

3.1. Índice de Jaccard

EL índice de Jaccard o coeficiente de Jaccard es una medida estadística que permite medir la similitud y diversidad de dos conjuntos, P y Q, sin importar la naturaleza de sus elementos [33]. Si bien podría decirse que la similitud entre 2 conjuntos podría calcularse a partir del tamaño (número cardinal) de la intersección, el número de elementos comunes por sí solo no puede decirnos cuán relativo es comparado con el tamaño de los conjuntos. De ahí la utilidad del coeficiente de Jaccard. Jaccard propone que, para medir la similitud, es necesario dividir el tamaño de la intersección por el tamaño de la unión para los dos conjuntos de datos.

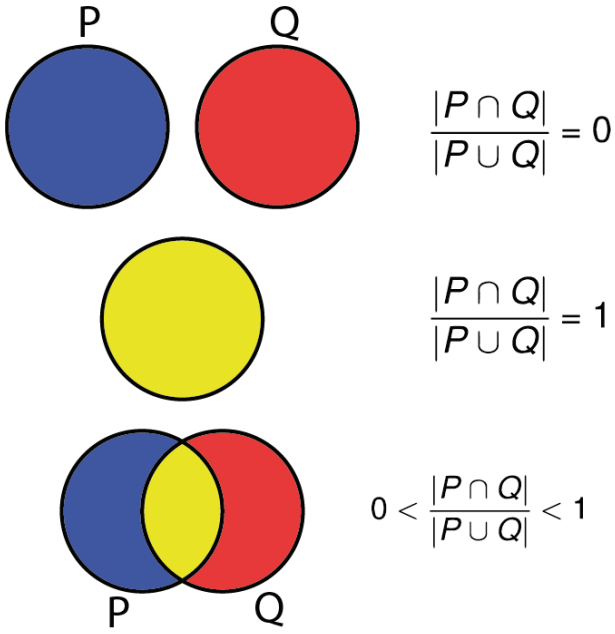


Figura 3.1: Índice de Jaccard.

Gráficamente podríamos describir los resultados de la similitud de Jaccard como se muestra en la figura 3.1. Si el valor obtenido es igual a 0, quiere decir que los 2 conjuntos son completamente disjuntos, por lo que no hay elementos en común, si el valor obtenido es 1, esto significa que ambos conjuntos son idénticos; por otro lado, si el valor se encuentra entre 0 y 1, significa que los conjuntos tienen un cierto número de elementos en común.

La distancia de Jaccard, $D(P,Q)$, es complementaria al coeficiente de Jaccard que mide la diferencia entre los conjuntos de datos (P y Q). Se calcula de la siguiente manera:

$$\begin{aligned} D(P,Q) &= 1 - J(P,Q) \\ &= 1 - \frac{|P \cap Q|}{|P \cup Q|} \\ &= \frac{|P \cup Q| - |P \cap Q|}{|P \cup Q|} \\ &= \frac{|P \cap Q| + |P - Q| + |Q - P| - |P \cap Q|}{|P \cap Q| + |P - Q| + |Q - P|} \\ &= \frac{|P - Q| + |Q - P|}{|P \cup Q|} \end{aligned} \tag{3.1}$$

En la figura 3.2, se muestra un ejemplo del uso del índice de Jaccard J para 2 conjuntos no binarios.

$$P = \{a,b,c\} \quad Q = \{a,b,c,d,e,f,g,h\}$$

$$J = \frac{|P \cap Q|}{|P \cup Q|} = \frac{|\{a,b,c\}|}{|\{a,b,c,d,e,f,g,h\}|} = \frac{3}{8}$$

Figura 3.2: Ejemplo del índice de Jaccard J.

Para el caso en el que los conjuntos correspondan a atributos binarios (verdadero o falso), la similitud con Jaccard se puede calcular de la siguiente forma:

$$J(P,Q) = \frac{c}{a+b+c}, \tag{3.2}$$

donde:

- **a:** es el número de atributos positivos solo en P.
- **b:** es el número de atributos positivos solo en Q .
- **c:** es el número de atributos positivos en ambos conjuntos simultáneamente.

Para esta tesis, el índice de Jaccard se utiliza para crear *clusters* de AP. Para esto, se utilizan algunos parámetros propios de los AP como son el canal, un valor de interferencia, la distancia relativa entre los 2 AP, un valor de probabilidad de enlace, obtenido con ayuda del modelo *Outage probability* [12], el valor de RSSI con el que un AP observa a otro, y un valor de calidad de enlace proporcionado por la tarjeta inalámbrica de cada AP.

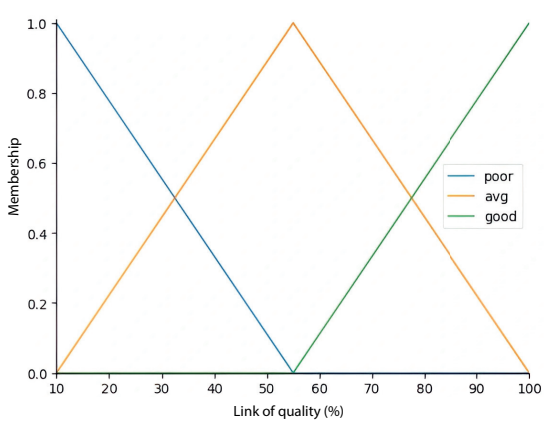
Debido a que la mayoría de los parámetros mencionados constan de un gran número de valores posibles, es necesario reducir estos campos de resultados a fin de que al momento de aplicar el índice de Jaccard los conjuntos comparados cuenten con valores en común y que de esta forma se puedan formar los *clusters*. Para reducir el número de valores posibles de estos parámetros se utilizó el concepto de lógica difusa, de este modo al fuzzificar los parámetros, los universos resultantes de cada parámetro se vuelven muy pequeños, lo que facilita que se encuentren coincidencias a la hora de comparar los conjuntos.

3.1.1. Fuzzificación de parámetros

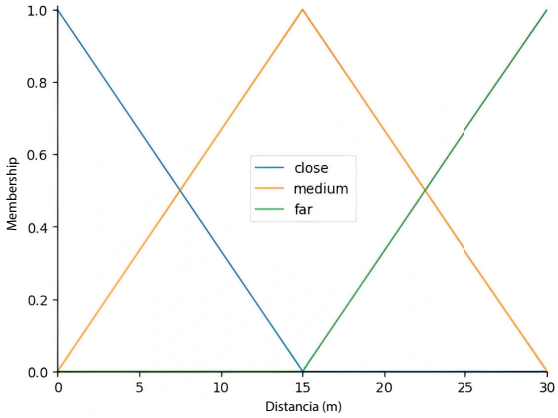
Como ya se ha explicado en el capítulo 2, el proceso de fuzzificación consiste en llevar valores numéricos a valores dentro de los conjuntos difusos, los cuales están formados por criterios preestablecidos para clasificar los valores de entrada.

En el caso de los parámetros obtenidos de los AP, con ayuda de la lógica difusa se creó el parámetro de *interferencia*, el cual depende de los valores de distancia y de la calidad de enlace entre 2 AP vecinos. De esta forma, los valores de distancia pasan de valores nítidos de entre 0 y 30 metros a un conjunto difuso que clasifica las distancias entre cerca, medio y lejos. Al mismo tiempo, el parámetro referente a la calidad de enlace pasa de porcentajes entre 0 y 100 a un conjunto difuso con los elementos pobre, media y buena.

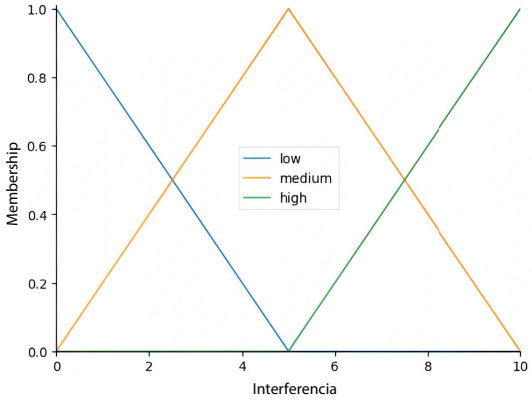
Con estos dos parámetros y por medio del proceso de inferencia difusa es que se obtiene el parámetro de interferencia (figura 3.3), en el que su conjunto difuso se tienen los valores de interferencia baja, media y alta.



(a) Función de membresía de calidad de enlace.



(b) Función de membresía de distancia.



(c) Función de membresía de calidad de interferencia.

Figura 3.3: Funciones de membresía.

Para el caso de los valores de RSSI, que se utilizan como valores de potencia de recepción, pueden estar entre -20 y $-90\,\mathrm{dBm}$, se fuzzifican para obtener solo 3 posibles salidas, bajo, medio o alto.

De esta forma, una vez obtenidos los nuevos valores de interferencia y RSSI para cada uno de los AP, se crean los conjuntos que serán comparados con el índice de Jaccard, con lo cual, cada conjunto finalmente tiene los siguientes elementos: canal, interferencia y RSSI. Una vez generados los clusters, en cada uno de estos se aplica el algoritmo de coloración. Para la creación de los clusters, se definió un valor J_u, que corresponde a un valor del índice de Jaccard entre 0 y 1, el cual se toma como umbral para la determinación de los clusters.

A partir de este umbral, se realizan comparaciones entre dos AP vecinos, si el índice de Jaccard resultante de la comparación de sus 2 conjuntos supera el umbral J_u, entonces formarán parte del cluster, en caso contrario, serán excluidos. El valor del umbral determina el tamaño del cluster, entre más cercano sea J_u a 1, el tamaño del cluster será más pequeño. Por el contrario, si J_u tiende a 0, entonces la mayoría de los AP pertenecerán a un solo cluster.

3.2. Algoritmo de coloración

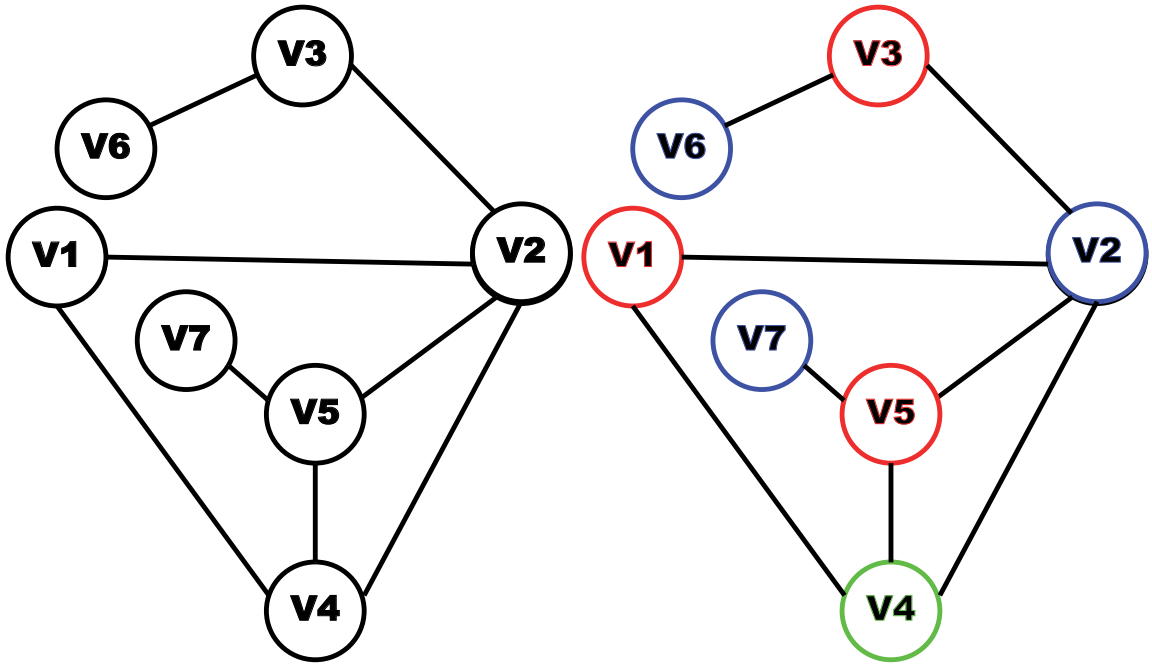
3.2.1. Algoritmo de coloración secuencial o voraz

El algoritmo secuencial se cataloga dentro del grupo de los algoritmos voraces. Esto quiere decir que mediante una metodología heurística elige la opción óptima en cada paso local con el fin de alcanzar una solución óptima general [24]. Para ello utiliza la matriz de adyacencias

del grafo. La designación de colores en el grafo se lleva a cabo de la siguiente manera, una vez ordenados los vértices por grados:

- Asignar el color 1 al primer vértice de la entrada del algoritmo.
- A continuación se escoge el siguiente vértice en el arreglo ordenado y se escoge el siguiente color pasible diferente respecto a sus vecinos. Este proceso se repite hasta que todos los vértices del grafo hayan sido coloreados.

En cada paso se va asignando un color a cada vértice, de ahí que el algoritmo sea considerado como voraz.



(a) Diagrama de grafo. (b) Coloración del grafo.

Figura 3.4: Algoritmo de coloración secuencial.

Finalmente, aplicando el algoritmo de coloración descrito, para el grafo de la figura 3.4 a), se obtienen la asignación mostrada en la figura 3.4 b).

3.2.2. Algoritmo de coloración Welsh- Powell

El algoritmo de Welsh-Powel (WP) es una derivación del algoritmo de coloración secuencial, con la diferencia de que ,en este caso, los vértices se ordenan de mayor a menor de acuerdo con su grado, es decir en función del número de vértices adyacentes.

Sea $V = v_1, v_2, v_3, ..., v_n$ el conjunto de vértices del grafo G, y sea $R = r_1, r_2, r_3, ..., r_4$ el conjunto de los 4 posibles colores. El algoritmo WP sigue los siguientes pasos:

1. Se calculan los grados (por grado se entiende al número de aristas que salen o entran de cada vértice y se añaden a un vector $Deg(v_i)$, donde $i = 1, 2, ..., n$.
2. Se selecciona un vértice no coloreado que tenga el grado más alto en el vector $Deg(v_i)$. Inicialmente, el primer color en el vector R es seleccionado como el color activo.
3. El vértice seleccionado es coloreado con el color activo. Posteriormente, se buscan aquellos vértices en la matriz de vecinos que no hayan sido coloreados y que a su vez no sean vecinos adyacentes del vértice seleccionado, y estos se colorean con el color activo actual.

4. Si aún existe un vértice en el grafo no coloreado, se selecciona el siguiente color en el vector R como el color activo y se regresa al Paso 2. De lo contrario, el programa ha terminado.

Para ejemplificar el funcionamiento del algoritmo, supongamos que se tiene la siguiente distribución de Puntos de Acceso (figura 3.5). Cada AP representa un vértice y cada canal de frecuencia representa un color. La representación de la distribución anterior en un grafo se muestra en la figura 3.6:

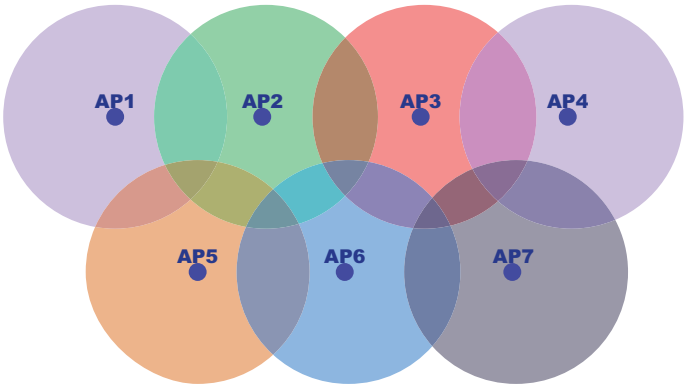


Figura 3.5: Representación de cobertura de AP.

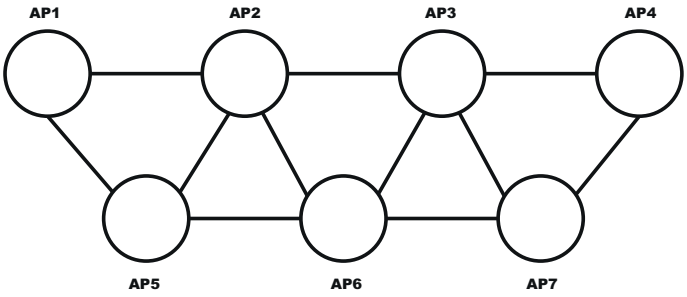


Figura 3.6: Representación del grafo de la red.

Siguiendo con los pasos del algoritmo, descritos anteriormente, el primer paso es calcular el grado de cada uno de los vértices. Suponiendo que el conjunto R es $R = \{rojo, verde, azul, negro\}$. El siguiente paso consiste en ver qué vértice será el primero en ser coloreado, para ello se elije el de mayor grado. Si existe más de un vértice con el mismo grado se elige de forma aleatoria. En este caso los vértices con mayor grado son AP2, AP3 y AP6. Para el ejemplo, se elige AP2, y a este se le asigna el primer color del vector R. Posteriormente, se colorea el vértice seleccionado junto con todos los vértices que no son adyacentes a él (rojo). Este paso se muestra en la figura 3.7.

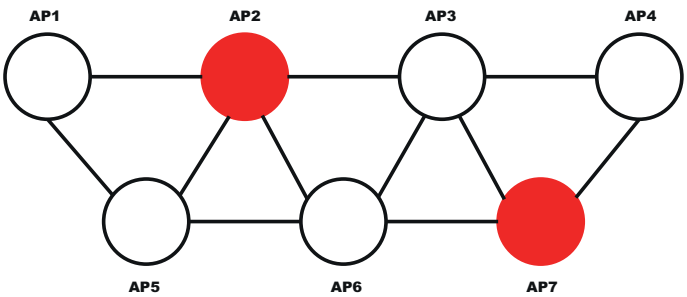


Figura 3.7: Paso 3 del algoritmo WP.

A continuación, se repite este último proceso y se va avanzando con los siguientes elementos del vector R, de tal forma que todos los vértices lleguen a estar coloreados sin que se repitan los colores entre vecinos adyacentes. Una vez aplicado por completo el algoritmo de Welsh-Powell, el grafo resultante se muestra en la figura 3.8.

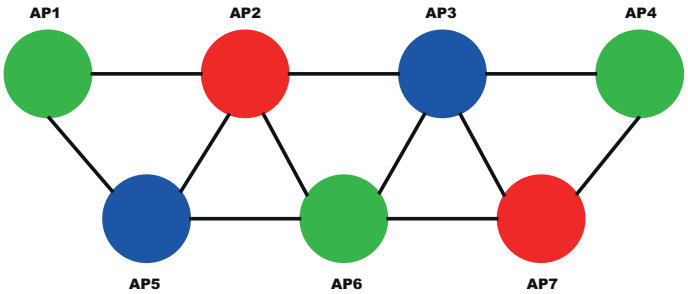


Figura 3.8: Representación del grafo resultante.

Aplicando este algoritmo a cada uno de los clusters obtenidos por medio del uso del índice de Jaccard descrito anteriormente, es como se da la asignación de los nuevos canales que utilizarán cada uno de los AP que conforman la red.

3.3. Implementación de mensajería entre *controlador* y *Access Point*

Parte importante del funcionamiento del protocolo está en el intercambio de información entre el controlador y todos los Access Point que conforman la red. Por medio de esta comunicación, es que se pueden intercambiar los datos necesarios de cada uno de los AP y de su vecindad con la cual funciona el algoritmo de asignación de colores. Este proceso de comunicación se da en dos vías: controlador-AP y AP-controlador. Para cada uno de los mensajes utilizados fue necesario hacer modificaciones dentro del codigo tanto en el controlador Ryu, como en *OpenvSwitch*. A continuación, se describen cada uno de los mensajes utilizados y las modificaciones realizadas.

En la figura 3.9 se muestra un diagrama con el intercambio de mensajes entre el controlador Ryu y los AP para la obtención de información. En esta figura se muestra la secuencia del intercambio de mensajes para la recolección de información y posteriormente el envío del nuevo canal para cada AP.

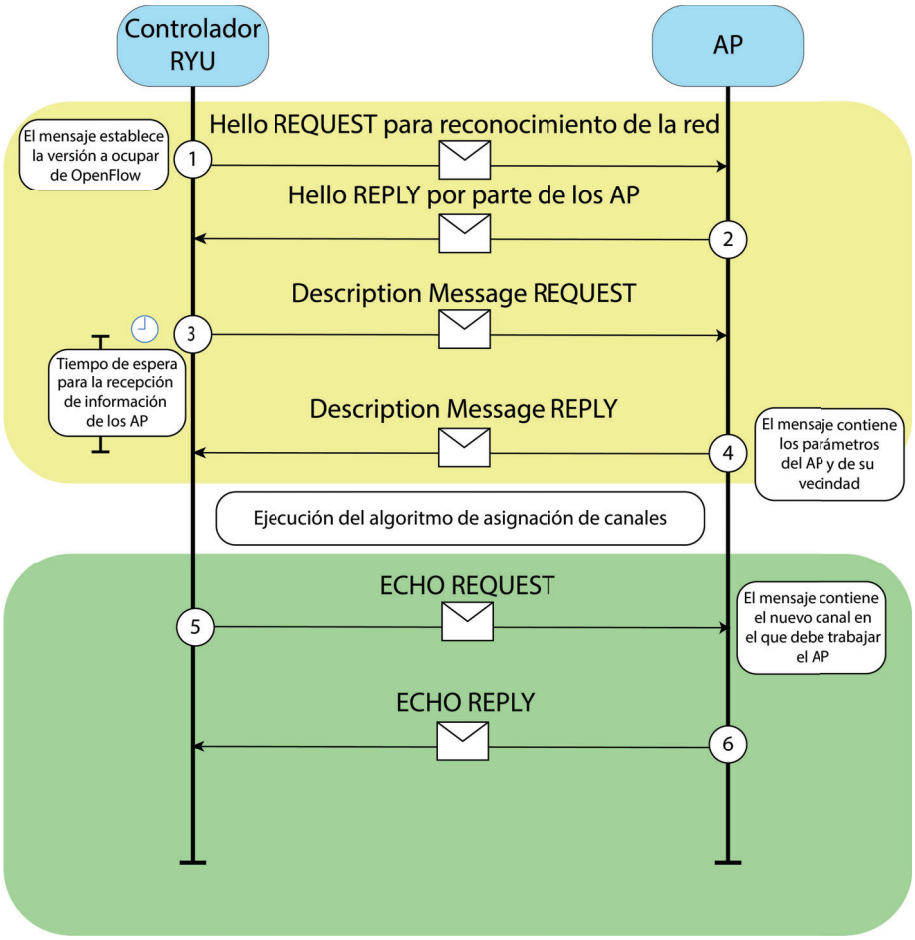


Figura 3.9: Flujo de mensajes entre controlador y AP.

3.3.1. Mensaje de recolección de información

Este mensaje sirve para obtener información sobre el estado de los parámetros de los Access Point, así como de sus vecinos. Esta información es usada por el algoritmo para la separación de *clusters*, y finalmente para la asignación de canales.

El mensaje ocupado para esta acción es el **Description Message**, descrito anteriormente en el capítulo 2. Originalmente, el mensaje de descripción tiene la estructura mostrada en la figura 3.10. Esta estructura se encuentra en el directorio `include/openflow`.

```
/* Body of reply to OFPMP_DESC request. Each entry is a NULL-terminated
* ASCII string. */
struct ofp_desc {
    char mfr_desc[DESC_STR_LEN]; /* Manufacturer description. */
    char hw_desc[DESC_STR_LEN]; /* Hardware description. */
    char sw_desc[DESC_STR_LEN]; /* Software description. */
    char serial_num[SERIAL_NUM_LEN]; /* Serial number. */
    char dp_desc[DESC_STR_LEN]; /* Human readable description of
    datapath. */
};
OFP_ASSERT(sizeof(struct ofp_desc) == 1056)
```

Figura 3.10: Estructura del mensaje de descripción.

Por medio de este mensaje, el controlador obtiene información sobre los parámetros de descripción de cada *switch* (AP). Para nuestro propósito, se añadieron más parámetros por medio de los cuales cada *Access Point* comunica al controlador la siguiente información:

- Nombre del *Access Point*.
- Potencia de transmisión en dBm.
- Canal actual .
- Lista de los AP vecinos asociados con su canal y el valor de RSSI visto por el AP en ese momento.

De esta manera, la estructura del mensaje ahora se ve de la siguiente forma:

```
/* Body of reply to OFPST_DESC request.  Each entry is a NULL-terminated
 * ASCII string. */
struct ofp_desc_stats {
    char mfr_desc[DESC_STR_LEN];       /* Manufacturer description. */
    char hw_desc[DESC_STR_LEN];        /* Hardware description. */
    char sw_desc[DESC_STR_LEN];        /* Software description. */
    char serial_num[SERIAL_NUM_LEN];   /* Serial number. */
    char dp_desc[DESC_STR_LEN];        /* Human readable description of
                                          the datapath. */
    char name_AP[SERIAL_NUM_LEN];      /*Name of the AP*/
    char type_switch[SERIAL_NUM_LEN];  /* Type of switch (1 or 0)*/
    char channel[SERIAL_NUM_LEN];      /* Channel*/
    char tx_power[SERIAL_NUM_LEN];     /*Transmition Power*/
    char neighbours[DESC_STR_LEN];     /*Neighgbours list*/

};
OFP_ASSERT(sizeof(struct ofp_desc_stats) == 1440);
```

Figura 3.11: Estructura modificada del mensaje de descripción.

Ya que este mensaje es de tipo *controlador-switch*, el controlador es el encargado de iniciar la comunicación, por medio de una petición `OFPST_DESC REQUEST`, con lo cual el switch responderá por medio de un `OFPST_DESC REPLY` enviando la información hacia el controlador.

3.3.1.1. Implementación del mensaje desde el controlador Ryu

Para que tanto el AP como el controlador puedan enviar y procesar esta nueva información (nombre del AP, potencia de transmisión, canal actual, lista de vecinos) es necesario modificar las estructuras y el comportamiento de las funciones que manejan este mensaje. Del lado del controlador, es necesario añadir los campos que recibirán la información enviada por los AP con el fin de que esta pueda ser decodificada y utilizada más adelante.

Como primer paso se modifica el archivo **ofproto_v1_3_parser.py** el cual se encuentra dentro del directorio `/usr/local/lib/python3.7/dist-packages/ryu/ofproto`, que contiene las funciones necesarias para la generación de diferentes peticiones `REQUEST` y `REPLY` para distintos tipos de mensajes asociados a la versión 1.3 de OpenFlow.

En este archivo, en la clase `OFPDescStats` la cual corresponde al mensaje de Descripción, se añaden los nuevos parámetros correspondientes a la nueva información que será recibida por el controlador. Al añadir los nuevos parámetros la clase queda de la siguiente manera:

```
class OFPDescStats(ofproto_parser.namedtuple('OFPDescStats', (
    'mfr_desc', 'hw_desc', 'sw_desc', 'serial_num', 'dp_desc','name_AP',
    'type_switch','channel','tx_power','neighbours'))):

_TYPE = {
    'ascii': [
        'mfr_desc',
        'hw_desc',
        'sw_desc',
        'serial_num',
        'dp_desc',
        'name_AP',
        'type_switch',
        'channel',
        'tx_power',
        'neighbours',
    ]
}

@classmethod
def parser(cls, buf, offset):
    desc = struct.unpack_from(ofproto.OFP_DESC_PACK_STR,
                              buf, offset)
    desc = list(desc)
    desc = [x.rstrip(b'\0') for x in desc]
    stats = cls(*desc)
    stats.length = ofproto.OFP_DESC_SIZE
    return stats
```

Figura 3.12: Clase correspondiente al mensaje de Descripción.

Finalmente, se deben de añadir el tamaño de cada uno de los nuevos parámetros que contendrán la estructura del mensaje, para que de esta forma el controlador pueda desempaquetar de forma correcta cada uno de los elementos que contiene el mensaje de respuesta.

Estos valores se añaden en el archivo **ofproto_v1_3.py** el cual contiene las descripciones de cada uno de los elementos que conforman los diferentes tipos de mensajes OpenFlow que pueden ser enviados o recibidos. El archivo se encuentra dentro del directorio `/usr/local/lib/python3.7/dist-packages/ryu/ofproto`.

Al añadir los valores asociados a los nuevos parámetros que tendrá el mensaje, el contenido del archivo queda de la siguiente manera:

```
......

# struct ofp_desc
DESC_STR_LEN = 256
DESC_STR_LEN_STR = str(DESC_STR_LEN)
SERIAL_NUM_LEN = 32
SERIAL_NUM_LEN_STR = str(SERIAL_NUM_LEN)
OFP_DESC_PACK_STR = '!' + \
                    DESC_STR_LEN_STR + 's' + \
                    DESC_STR_LEN_STR + 's' + \
                    DESC_STR_LEN_STR + 's' + \
                    SERIAL_NUM_LEN_STR + 's' + \
                    DESC_STR_LEN_STR + 's' + \
                    SERIAL_NUM_LEN_STR + 's' + \
                    SERIAL_NUM_LEN_STR + 's' + \
                    SERIAL_NUM_LEN_STR + 's' + \
                    SERIAL_NUM_LEN_STR + 's' + \
                    DESC_STR_LEN_STR + 's'

OFP_DESC_SIZE = 1440
assert calcsize(OFP_DESC_PACK_STR) == OFP_DESC_SIZE
......
```

Figura 3.13: Definición del tamaño del mensaje de Descripción.

Para que el controlador pueda decodificar el mensaje de forma correcta, se debe de tener en cuenta el tamaño del mensaje, al añadir los nuevos parámetros el tamaño del mensaje es de 1440 bytes, este valor debe corresponder tanto del lado de OpenvSwitch, como del controlador, como se muestra en los códigos de las figuras 3.11 y 3.13.

3.3.1.2. Implementación del mensaje de descripción en OpenvSwitch

El mensaje de respuesta con la información solicitada se genera en OpenvSwitch. Para rellenar estos campos es necesario obtener la información del AP y de su vecindad. Esta información se genera una vez que se recibe el `REQUEST` que viene del controlador.

Para la implementación de este mensaje de respuesta se realizaron las siguientes modificaciones dentro de los archivos de OpenvSwitch:

- En el archivo **openflow-common.h**, que se encuentra dentro del directorio `include/openflow` de `openvswitch-2.13.4`, se añadieron los nuevos parámetros a la estructura del mensaje de descripción correspondientes a la información solicitada por el controlador. Una vez modificado este archivo la estructura del mensaje queda como se mostró en la figura 3.11.

- Para que el mensaje se pueda visualizar de forma correcta en consola al momento de realizar un debug o utilizar un visualizador de paquetes, debemos de modificar la función `ofp_print_ofpst_desc_reply`, que se encarga de crear la descripción del cuerpo del mensaje, esta función se encuentra en el archivo `/lib/ofp-print.c`. Al añadir la descripción de los nuevos parámetros, la función queda de la siguiente manera:

```
static enum ofperr
ofp_print_ofpst_desc_reply(struct ds *string, const struct ofp_header *oh)
{
    const struct ofp_desc_stats *ods = ofpmsg_body(oh);

    ds_put_char(string, '\n');
    ds_put_format(string, "Manufacturer: %.*s\n",
            (int) sizeof ods->mfr_desc, ods->mfr_desc);
    ds_put_format(string, "Hardware: %.*s\n",
            (int) sizeof ods->hw_desc, ods->hw_desc);
    ds_put_format(string, "Software: %.*s\n",
            (int) sizeof ods->sw_desc, ods->sw_desc);
    ds_put_format(string, "Serial Num: %.*s\n",
            (int) sizeof ods->serial_num, ods->serial_num);
    ds_put_format(string, "DP Description: %.*s\n",
            (int) sizeof ods->dp_desc, ods->dp_desc);

    ds_put_format(string, "Type_Switch: %.*s\n",
            (int) sizeof ods->type_switch, ods->type_switch);
    ds_put_format(string, "Wifi Channel: %.*s\n",
            (int) sizeof ods->channel, ods->channel);
    ds_put_format(string, "Name AP: %.*s\n",
            (int) sizeof ods->name_AP, ods->name_AP);
    ds_put_format(string, "Tx Power: %.*s\n",
            (int) sizeof ods->tx_power, ods->tx_power);
    ds_put_format(string, "Neighbours: %.*s\n",
            (int) sizeof ods->neighbours, ods->neighbours);
    return 0;
}
```

Figura 3.14: Función para la impresión en consola del mensaje de Descripción.

- Además de añadir las nuevas variables a la estructura propia del mensaje, también se deben añadir a la estructura de los parámetros generales de Openflow llamada `ofproto`. Esta se encuentra en el archivo `/ofproto/ofproto-provider.h`

```
struct ofproto {
    struct hmap_node hmap_node; /* In global 'all_ofprotos' hmap. */
    const struct ofproto_class *ofproto_class;
    char *type;                 /* Datapath type. */
    char *name;                 /* Datapath name. */

    /* Settings. */
    uint64_t fallback_dpid;     /* Datapath ID if no better choice found. */
    uint64_t datapath_id;       /* Datapath ID. */
    bool forward_bpdu;          /* Option to allow forwarding of BPDU frames
                                 * when NORMAL action is invoked. */
    char *mfr_desc;             /* Manufacturer (NULL for default). */
    char *hw_desc;              /* Hardware (NULL for default). */
    char *sw_desc;              /* Software version (NULL for default). */
    char *serial_desc;          /* Serial number (NULL for default). */
    char *dp_desc;              /* Datapath description (NULL for default). */

    char *name_AP;
    char *channel;
    char *type_switch;
    char *tx_power;
    char *neighbours;

    ......
```

Figura 3.15: Estructura ofproto.

Estas variables añadidas a la estructura `ofproto` mostradas en la figura 3.15, son usadas para guardar la información del AP y de su vecindad para que posteriormente sean enviadas al controlador.

En el mismo archivo en donde se encuentra la estructura `ofproto`, se creó una pequeña estructura llamada **Tuple** mostrada en la figura 3.16 con variables que servirán para guardar de manera temporal la información requerida por el mensaje y que sera leída por un archivo de texto que se explica más adelante.

```
struct  Tuple {
    char channel[25];
    char tx_power[25];
    char neighbours[256];
    };
```

Figura 3.16: Estructura Tuple.

- Finalmente, dentro del archivo `/ofproto/ofproto.c`, es donde se realizan los cambios más significativos para obtener la información y enviar el mensaje hacia el controlador. Dentro de este archivo lo primero es inicializar las variables de la estructura `ofproto` agregadas en el paso anterior. De esta forma, en la sección de inicialización, las variables quedan de la siguiente forma:

```
/* Initialize. */
ovs_mutex_lock(&ofproto_mutex);
memset(ofproto, 0, sizeof *ofproto);
ofproto->ofproto_class = class;
ofproto->name = xstrdup(datapath_name);
ofproto->type = xstrdup(datapath_type);
hmap_insert(&all_ofprotos, &ofproto->hmap_node,
            hash_string(ofproto->name, 0));
ofproto->datapath_id = 0;
ofproto->forward_bpdu = false;
ofproto->fallback_dpid = pick_fallback_dpid();
ofproto->mfr_desc = NULL;
ofproto->hw_desc = NULL;
ofproto->sw_desc = NULL;
ofproto->serial_desc = NULL;
ofproto->dp_desc = NULL;


ofproto->channel= NULL;
ofproto->type_switch=NULL;
ofproto->name_AP=NULL;
ofproto->tx_power=NULL;
ofproto->neighbours=NULL;
...
...
```

Figura 3.17: Inicialización de variables.

Después, añadiremos las variables a la función `ofproto_destroy__`, en la cual una vez usadas las variables para la creación del mensaje, son destruidas para liberar espacio en la memoria. La sección de la función en la que son agregadas las variables queda de la siguiente forma:

```
static void
ofproto_destroy__(struct ofproto *ofproto)
    OVS_EXCLUDED(ofproto_mutex)
{
    struct oftable *table;

    cmap_destroy(&ofproto->groups);

    ovs_mutex_lock(&ofproto_mutex);
    hmap_remove(&all_ofprotos, &ofproto->hmap_node);
    ovs_mutex_unlock(&ofproto_mutex);

    free(ofproto->name);
    free(ofproto->type);
    free(ofproto->mfr_desc);
    free(ofproto->hw_desc);
    free(ofproto->sw_desc);
    free(ofproto->serial_desc);
    free(ofproto->dp_desc);

    free(ofproto->channel);
    free(ofproto->type_switch);
    free(ofproto->name_AP);
    free(ofproto->tx_power);
    free(ofproto->neighbours);
    ...
    ...
```

Figura 3.18: Función ofproto_destroy.

- Como última modificación, en la función `handle_desc_stats_request`, es en donde se realiza la recolección de información necesaria para crear el mensaje, así como el empaquetamiento del mismo para después ser enviado.

```
static enum ofperr
handle_desc_stats_request(struct ofconn *ofconn,
                          const struct ofp_header *request)
{
    static const char *default_mfr_desc = "Nicira, Inc.";
    static const char *default_hw_desc = "Open vSwitch";
    static const char *default_sw_desc = VERSION;
    static const char *default_serial_desc = "None";
    static const char *default_dp_desc = "None";

    system("sudo python3 /home/pi/openvswitch-2.13.4/ofproto/cmd.py");
    system("sudo python3 /home/pi/openvswitch-2.13.4/ofproto/ap_data.py");

    struct  Tuple params;
    FILE *f;
    f = fopen("/home/pi/openvswitch-2.13.4/ofproto/data.txt", "r");
    fscanf(f, "%s",params.channel);
    fscanf(f, "%s", params.tx_power);
    fclose(f);



   f = fopen("/home/pi/openvswitch-2.13.4/ofproto/neighbours.txt", "r");
   fscanf(f, "%s",params.neighbours);
   fclose(f);


    char *default_name_AP="AP13";
    char *default_type_switch="1";
    char *default_channel = params.channel;
    char *default_tx_power=params.tx_power;
    char *default_neighbours=params.neighbours;

    ...
    ...
```

Figura 3.19: Función ofproto_destroy.

El código de la figura 3.19 muestra la primera parte de la función en la cual se definen los valores iniciales de las diferentes variables que conforman al mensaje de descripción. Respecto a los valores de las variables correspondientes a la información solicitada, estos son obtenidos de los archivos de texto `data.txt` y `neighbours.txt` que a su vez son generados por dos scripts de Python, `cmd.py` y `ap_data.py` (ver Apéndice D), los cuales son ejecutados por la función ya mencionada a través del comando **system**. Una vez leídos los archivos de texto, la información extraída se guarda en las variables temporales de la estructura *Tuple*, descrita anteriormente, para después pasar esta información a las variables de la estructura `ofproto`, con las cuales se formará el mensaje a enviar.

En la segunda parte de la función se empaquetan cada uno de las variables que forman parte del mensaje, como se muestra en el código de la figura 3.20, una vez que todas son empaquetadas, se envía el mensaje hacia el controlador.

```
.......
.......
struct ofproto *p = ofconn_get_ofproto(ofconn);
struct ofp_desc_stats *ods;
struct ofpbuf *msg;
    msg = ofpraw_alloc_stats_reply(request, 0);
ods = ofpbuf_put_zeros(msg, sizeof *ods);
ovs_strlcpy(ods->mfr_desc, p->mfr_desc ? p->mfr_desc : default_mfr_desc,
            sizeof ods->mfr_desc);
ovs_strlcpy(ods->hw_desc, p->hw_desc ? p->hw_desc : default_hw_desc,
            sizeof ods->hw_desc);
ovs_strlcpy(ods->sw_desc, p->sw_desc ? p->sw_desc : default_sw_desc,
            sizeof ods->sw_desc);
ovs_strlcpy(ods->serial_num,
            p->serial_desc ? p->serial_desc : default_serial_desc,
            sizeof ods->serial_num);
ovs_strlcpy(ods->dp_desc, p->dp_desc ? p->dp_desc : default_dp_desc,
            sizeof ods->dp_desc);

ovs_strlcpy(ods->channel, p->channel ? p->channel : default_channel,
            sizeof ods->channel);
ovs_strlcpy(ods->type_switch, p->type_switch ? p->type_switch :
            default_type_switch, sizeof ods->type_switch);
ovs_strlcpy(ods->name_AP, p->name_AP ? p->name_AP : default_name_AP,
            sizeof ods->name_AP);
ovs_strlcpy(ods->tx_power, p->tx_power ? p->tx_power : default_tx_power,
            sizeof ods->tx_power);
ovs_strlcpy(ods->neighbours, p->neighbours ? p->neighbours : default_neighbours,
            ~sizeof ods->neighbours);
ofconn_send_reply(ofconn, msg);
return 0;
}
```

Figura 3.20: Función handle_desc_stats_request.

3.3.2. Mensaje de asignación de canal ECHO

Una vez aplicado el algoritmo de asignación de canal, se debe de comunicar a cada uno de los AP de la red cuál será el nuevo canal en el que deben de trabajar, para esto, el controlador, envía un mensaje a cada AP con el nuevo canal asignado. Esta comunicación se da mediante el mensaje `ECHO`. Este es un mensaje simétrico, esto significa que el mensaje puede ser enviado sin solicitud previa, en cualquier dirección. En este caso el mensaje es enviado por el controlador. Además, de poder mandar la información que se requiera ya sea al controlador o a un *switch*, el mensaje `ECHO`, también sirve para verificar el estado de vida de una conexión entre el controlador y un *switch*, también puede ser usado para medir la latencia y el ancho de banda del enlace.

3.3.2.1. Implementación del mensaje en el controlador

Para establecer el envío de un mensaje `ECHO`, basta con usar las funciones proporcionadas por la API de Openflow [34], y añadir el nuevo canal del AP en el parámetro `data`.

```
def send_echo_request(self, datapath, data):
ofp_parser = datapath.ofproto_parser

req = ofp_parser.OFPEchoRequest(datapath, data)
datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPEchoRequest,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_request_handler(self, ev):
    self.logger.debug('OFPEchoRequest received: data=%s',
                      utils.hex_array(ev.msg.data))
```

Figura 3.21: Funciones para enviar un mensaje ECHO.

3.3.2.2. Implementación del mensaje en OpenVSwitch

Del lado de los AP, una vez que se recibe el mensaje `ECHO`, que se envía desde el controlador, se debe desempaquetar el canal enviado en el apartado de `data`, para después realizar el cambio en el propio AP.

Para esto, dentro del archivo `\lib\ofp-util.c`, es donde se manipula la información recibida del mensaje para realizar el cambio de canal. Especificamente, es en la función `ofputil_encode_echo_request` donde se recibe y decodifica el mensaje `ECHO`.

```
/* Creates and returns an OFPT_ECHO_REPLY message matching the
 * OFPT_ECHO_REQUEST message in 'rq'. */
struct ofpbuf *
ofputil_encode_echo_reply(const struct ofp_header *rq)
{
    struct ofpbuf rq_buf = ofpbuf_const_initializer(rq, ntohs(rq->length));
    ofpraw_pull_assert(&rq_buf);

    struct ofpbuf *reply = ofpraw_alloc_reply(OFPRAW_OFPT_ECHO_REPLY,
                                              rq, rq_buf.size);
    ofpbuf_put(reply, rq_buf.data, rq_buf.size);

    sleep(5);

    char channel[32];
    system("sudo python3 /home/pi/openvswitch-2.13.4/ofproto/cmd.py");

    FILE *f;
    f = fopen("/home/pi/openvswitch-2.13.4/ofproto/data.txt", "r");
    fgets(channel,32,f);
    fclose(f);

    int c = atoi(rq_buf.data);
    int ch =atoi(channel);
    char buf[256];

    if(c != ch){
        sprintf(buf," python3 /home/pi/openvswitch-2.13.4/lib/change_channel.py
            %d", c);
        system(buf);
     }

    return reply;
}
```

Figura 3.22: Función `ofputil_encode_echo_reply`.

Una vez que que se recibe el mensaje y se desempaquetan los datos del mismo, se obtiene el canal actual en el que se encuentra el AP, con el fin de compararlo con el nuevo canal seleccionado por el controlador. En caso de que se trate del mismo canal, no se llevará a cabo el cambio. De esta forma se evita que el canal se actualice de forma continua, y provoque un corte constate de comunicación entre los dispositivos conectados y el AP. Si el canal recibido es diferente al que tienen el AP en ese momento, entonces se realiza la actualización del canal. Para realizar este cambio se debe de modificar el archivo que contiene los parámetros generales del AP, este archivo se describe en el apéndice C.

La modificación del archivo se da mediante un pequeño script en python llamado `change_channel.py` (ver Apéndice D), el cual realiza la modificación en el archivo y reinicia el AP para que este pueda asimilar el cambio de canal.

Capítulo 4

Implementación y resultados

En este capítulo, se describe la implementación física de la propuesta, así como un conjunto de simulaciones bajo el mismo escenario con el objetivo de mostrar los beneficios del algoritmo de coloración combinado con la métrica de Jaccard y la lógica difusa. Respecto a los resultados mostrados, se realiza la comparación entre una asignación de canales aleatoria y la asignación de frecuencias a partir del algoritmo propuesto. Para evaluar el desempeño de ambos mecanismos de asignación se utilizaron las métricas de *throughput* y *jitter*.

- ***Throughput*:** Representa la tasa promedio de entrega de paquetes exitosos sobre un canal por unidad de tiempo. Este parámetro se mide en bits por segundo [bps].

- ***Jitter*:** También se conoce técnicamente como variación del retardo de los paquetes. Esto se refiere a la variación de la demora de tiempo entre los paquetes de datos a través de una red.

Como se ha mencionado durante este trabajo de tesis, el propósito es implementar el algoritmo de asignación de canales propuesto dentro de un escenario real con el fin de obtener datos que muestren el comportamiento real de una red y de esta forma verificar que tenga mejoras en el *throughput* cuando en esta se tienen fenómenos como multitrayectorias, atenuación de la señal, fluctuaciones lentas, e interferencia, los cuales son muy difíciles de emular en los emuladores existentes como Mininet [16].

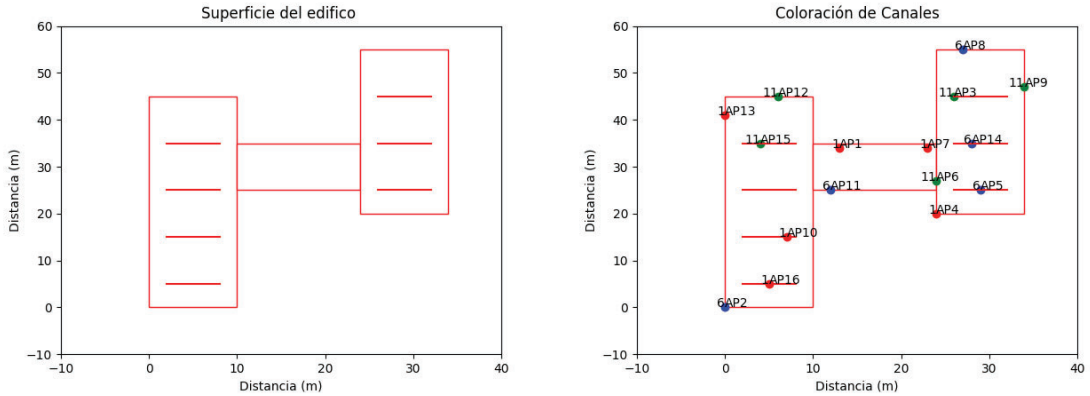
4.1. Construcción de la red

Para probar el funcionamiento del algoritmo, se implementó una red física que consta de 16 Raspberry Pi que funcionan como *Access Point*, 2 *switches* de 8 puertos cada uno, una computadora en la cual se estableció el controlador de la red y 13 Raspberry Pi que funcionan como clientes de la red.

En cuanto a la distribución de la red se consideró la azotea del edificio Luis G. Valdés Vallejo, de la Facultad de Ingeniería en Ciudad Universitaria. El cual tiene una longitud de $45\,\mathrm{m}$ por $35\,\mathrm{m}$, éste se muestra en la figura 4.1.

Figura 4.1: Edificio Luis G. Valdés Vallejo.

Para establecer la posición de cada uno de los AP se mapeó la superficie del edificio, como se muestra en la figura 4.2 a) y se generaron varias distribuciones de puntos para colocar los AP. Finalmente, se optó por la distribución de la figura 4.2 b), debido a que se observó una distribución más homogénea en cuanto a las distancias de separación entre los AP, además de abarcar la mayor parte de la superficie disponible para el despliegue de la red.



(a) Mapeo de la superficie del edificio

(b) Posicionamiento de los AP

Figura 4.2: Despliegue de la red en la azotea del Valdés Vallejo.

En la figura anterior, cada AP se muestra de un color según sea el canal en el que se encuentra en ese momento, el número del canal se puede visualizar a la izquierda del nombre de cada AP. Una vez establecida la distribución de la red física, se construyó la red con todos los equipos físicos necesarios para probar el algoritmo diseñado.

4.2. Escenario físico

La distribución utilizada para la construcción de la red fue la mostrada en la figura 4.2 b), añadiendo los dispositivos que funcionan como clientes el escenario final queda como se muestra en la figura 4.3, donde estos se muestran en color naranja.

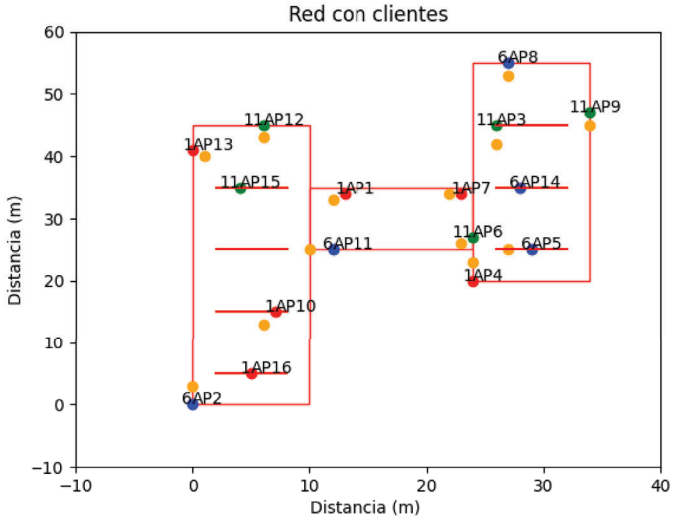


Figura 4.3: Red implementada con clientes.

A partir de este escenario, se realizaron las pruebas de *throughput* y *jitter*, tanto para el caso en el que se tiene una asignación aleatoria, como para la asignación obtenida a partir de la implementación del algoritmo. El primer grupo de pruebas se realizó sobre la red mostrada en la figura 4.3, la cual corresponde a la asignación aleatoria. Una vez obtenido este primer conjunto de pruebas se aplicó el algoritmo propuesto. La formación de *clusters* se muestra en la figura 4.4 utilizando el índice de Jaccard.

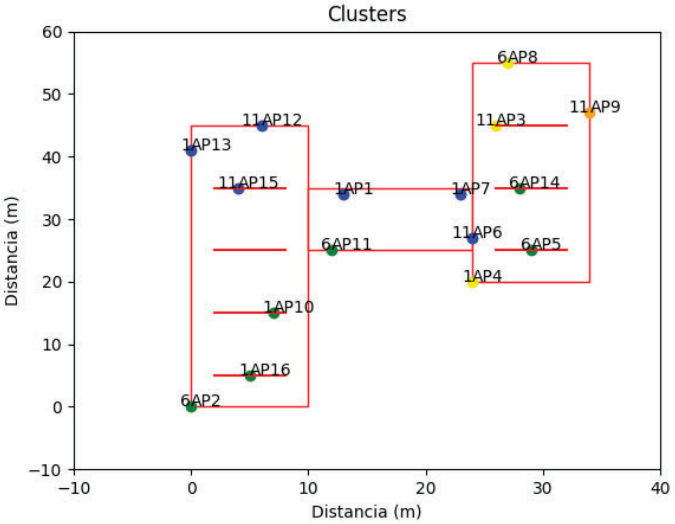


Figura 4.4: Clusters generados con el índice de Jaccard.

Para el experimento se utilizó un valor de 0.6 correspondiente al índice de Jaccard. Este

valor, se escogió después de realizar pruebas de generación de clusters para diferentes escenarios, variando el índice entre valores de 0 y 1. Como resultado de estas pruebas, se observó el número promedio de elementos por clusters que se obtuvieron al momento de realizar la agrupación de APs. De esta manera se llegó a la conclusión de que, al utilizar un valor de 0.6, los elementos de los clusters oscilan entre 4 y 6 elementos en promedio. Esto es conveniente ya que permite una buena distribución de los canales al momento de aplicar el algoritmo de coloración.

Una vez obtenidos los clusters, se aplica el algoritmo de coloración de Welsh-Powell a cada uno de ellos. Como resultado, se obtiene una nueva asignación de canales mostrada en la figura 4.5.

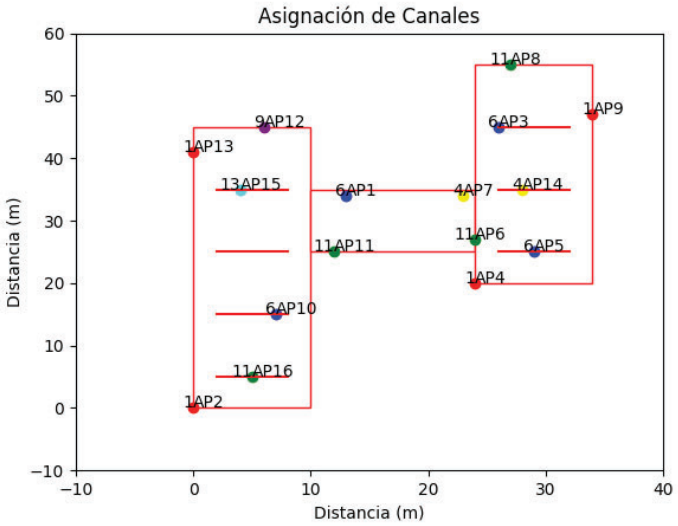


Figura 4.5: Asignación de nuevos canales.

A partir de esta nueva asignación de canales en la red se realizó el segundo conjunto de pruebas para medir el *throughput* y *jitter* en la red.

4.3. Pruebas y resultados

Para medir los parámetro de *throughput* y *jitter*, se hizo uso de la herramienta *iperf3*. Para el caso del *throughput* se utilizó el protocolo TCP. Concretamente, las pruebas consistieron en generar tráfico con los 13 clientes disponibles en la red, donde el tráfico TCP generado por los clientes va dirigido a dos *host* correspondientes a 2 máquinas virtuales creadas con XEN, las cuales se encuentran en la misma máquina en la que está el controlador Ryu. Estos *host* funcionan como servidores a los cuales todos los clientes de la red pueden acceder. Mientras se está generando el trafico, se realizó la medición del *throughput* por un periodo de 240 segundos. Mediante un conjunto de pruebas, se observó que la red necesita un periodo de 60 s para que los clientes de la red puedan asociarse a su AP (después de que al algoritmo de colores les asigna una frecuencia por medio del controlador), mientras que los restantes 3 minutos corresponden a la lectura de una transmisión constante por parte de los clientes. Esta medición se realizó con ayuda de uno de los 13 clientes disponibles en la red. El cliente seleccionado es el encargado de ejecutar la herramienta *iperf3*, mientras que el servidor se ejecuta en uno de los *hosts* ya mencionados. De esta forma, se envía tráfico de ida y vuelta. La capacidad del canal se estableció en 10 Mbps. Este experimento se realizó 20 veces y a partir de este conjunto de experimentos se promediaron los valores obtenidos en cada segundo.

Respecto al parámetro de *jitter* se realizó el mismo número de pruebas que con el parámetro anterior. En este caso, el protocolo utilizado corresponde a UDP. Para este conjunto de prue-

bas también se fijo una capacidad del canal de $10\,\text{Mbps}$. La prueba se midió en milisegundos a lo largo del mismo periodo de tiempo (240 segundos). Nuevamente, se consideró la asignación de canales aleatoria y la asignación producto de la aplicación del algoritmo propuesto.

Finalmente, en la tabla 4.1 se muestran los rangos de valores que utiliza el algoritmo para los procesos de fuzzificación de las entradas numéricas, el valor del índice de Jaccard utilizado para la creación de los clusters, entre otros parámetros.

Variable	Nombre	Valor o rango
J_U	Índice de Jaccard	0.6
lq	Calidad de enlace	Valores numéricos: 0 a 100 %
		Conjuntos difuso: pobre, promedio, bueno
d	Distancia	Valores numéricos: 0 a 30 m
		Conjunto difuso: cerca, medio, lejos
I	Interferencia	Valores numéricos: 0 a 10
		Conjunto difuso: baja, media, alta
RSSI	Received Signal Strength Indicator	Valores numéricos: -10 a -90 dBm
		Conjunto difuso: baja, media, alta

Tabla 4.1: Tabla de parámetros del algoritmo diseñados.

En la figura 4.6, se muestra la comparación del *throughput* obtenido para el algoritmo aleatorio, así como la propuesta de esta tesis. La línea verde corresponden a el resultado de la asignación realizada con ayuda del algoritmo propuesto, mientras que la línea amarilla corresponde a la asignación de canales aleatoria, también se muestra el promedio para ambos escenarios. En esta figura se puede observar que existe una mejora en el *throughput* de la red. Si se compara el *throughput* promedio obtenido en ambos escenarios,(lineas rectas constantes), se puede observar que el algoritmo propuesto (linea verde), alcanza una mejora de un $330.77\,\%$ en este parámetro con respecto a una asignación aleatoria (linea amarilla).

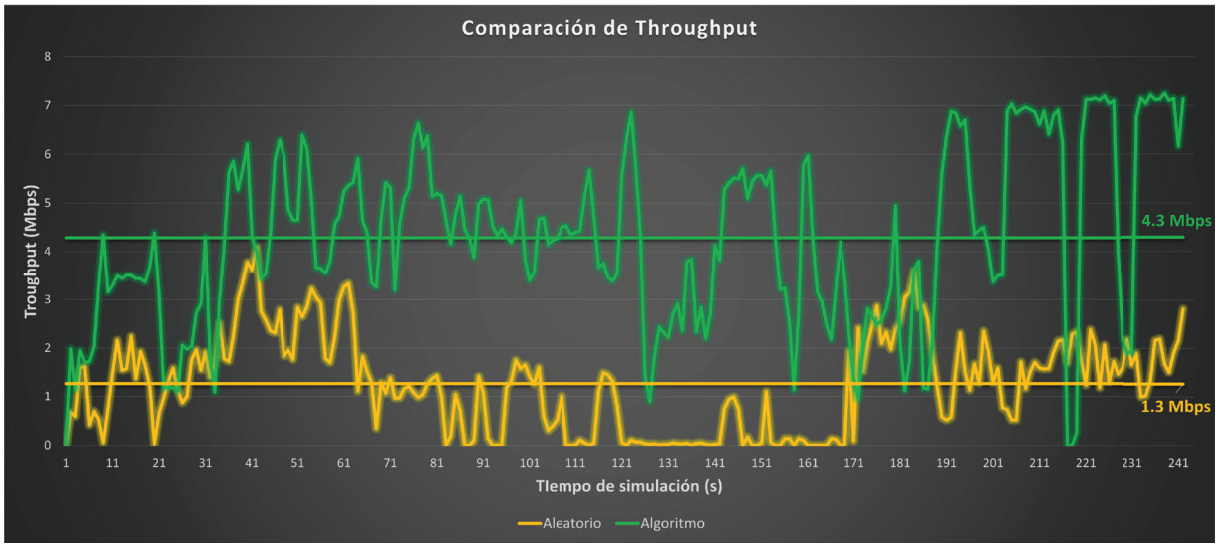


Figura 4.6: Resultados del *Throughput*.

Los resultados del *jitter* se pueden ver en la figura 4.7. Las lineas verdes corresponden a los resultados del algoritmo propuesto, mientras las líneas azules corresponden a los resultados de la asignación aleatoria de los canales. Las líneas rectas representan el promedio para ambos algoritmos.

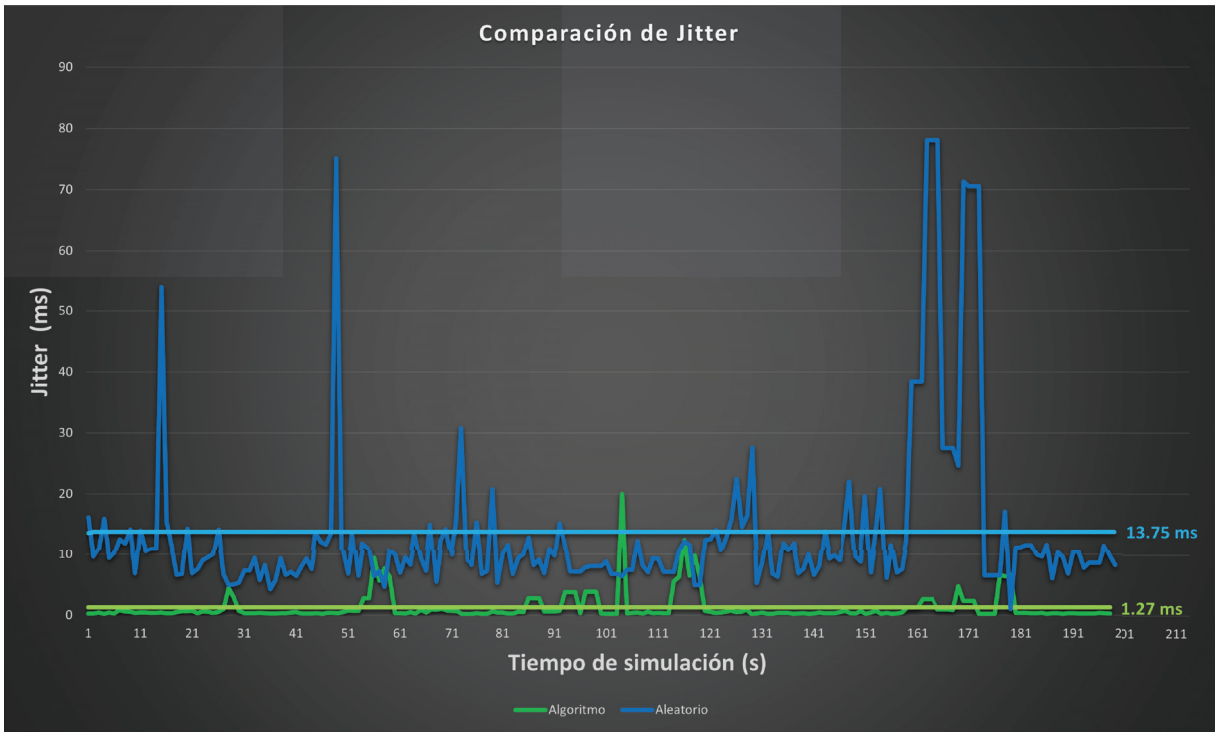


Figura 4.7: Resultados del *Jitter*.

Al igual que con el *throughput*, en este parámetro se puede observar una mejora significativa. En este caso el *jitter* promedio obtenido para el algoritmo propuesto (linea continua verde) es casi 11 veces más pequeño, que en una asignación aleatoria (linea continua azul). Esto significa que el retraso en el envío de paquetes sobre la red es más pequeño que bajo un esquema aleatorio, lo que implica un mejor enlace entre un cliente y un AP al momento de estar transmitiendo información.

A partir de los resultados observados con los parámetros de *throughput* y *jitter*, se puede afirmar que el algoritmo diseñado, proporciona grandes mejorías en la red, minimizando la interferencia generada por los AP vecinos dentro de la red y reduciendo el número de colisiones entre paquetes.

4.4. Emulación

De forma complementaria, se realizó la emulación del mismo escenario físico con ayuda del emulador Mininet WiFi [35], el cual permite generar redes SDN virtuales, teniendo la posibilidad de añadir estaciones WiFi virtualizadas (STA) y puntos de acceso (AP), ya sean estáticos o móviles. Para ello, se creó el escenario planteado en la figura 4.3, con el propósito de comparar el comportamiento de los resultados de los parámetros de *throughput* y *jitter* y analizar si existen cambios en los resultados obtenidos entre un escenario real y un escenario simulado. En el caso del escenario simulado se añadieron 2 *host* a la red creada los cuales actuarán como los *host* creados con XEN en el escenario físico, para que de esta forma se puedan realizar las pruebas de medición de la misma forma que en el escenario real.

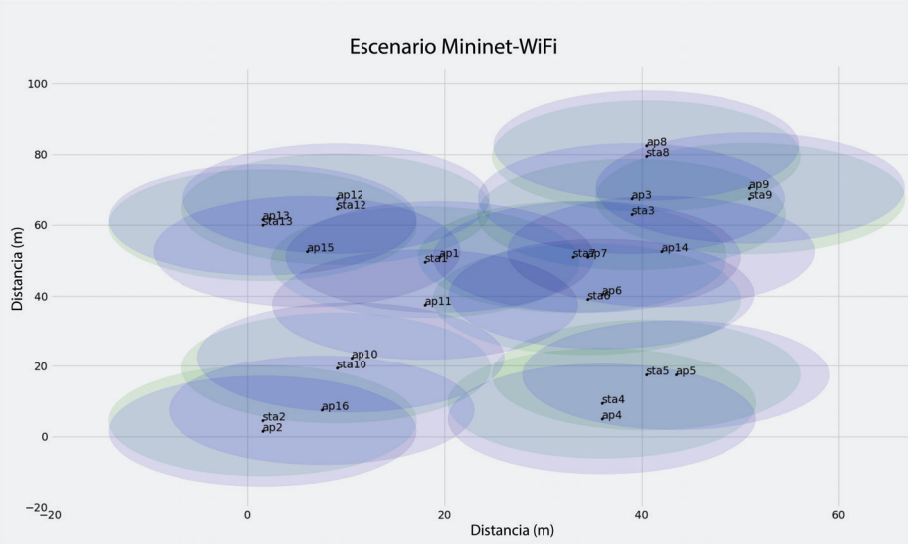


Figura 4.8: Escenario Simulado.

En la figura 4.8 se muestra la distribución de los AP y los clientes que conforman la red. Para medir los parámetros de *throughput* y *jitter*, se realizaron los mismos conjuntos de pruebas que en el escenario físico con el fin de replicar las condiciones y poder realizar una comparación entre los resultados de los dos escenarios.

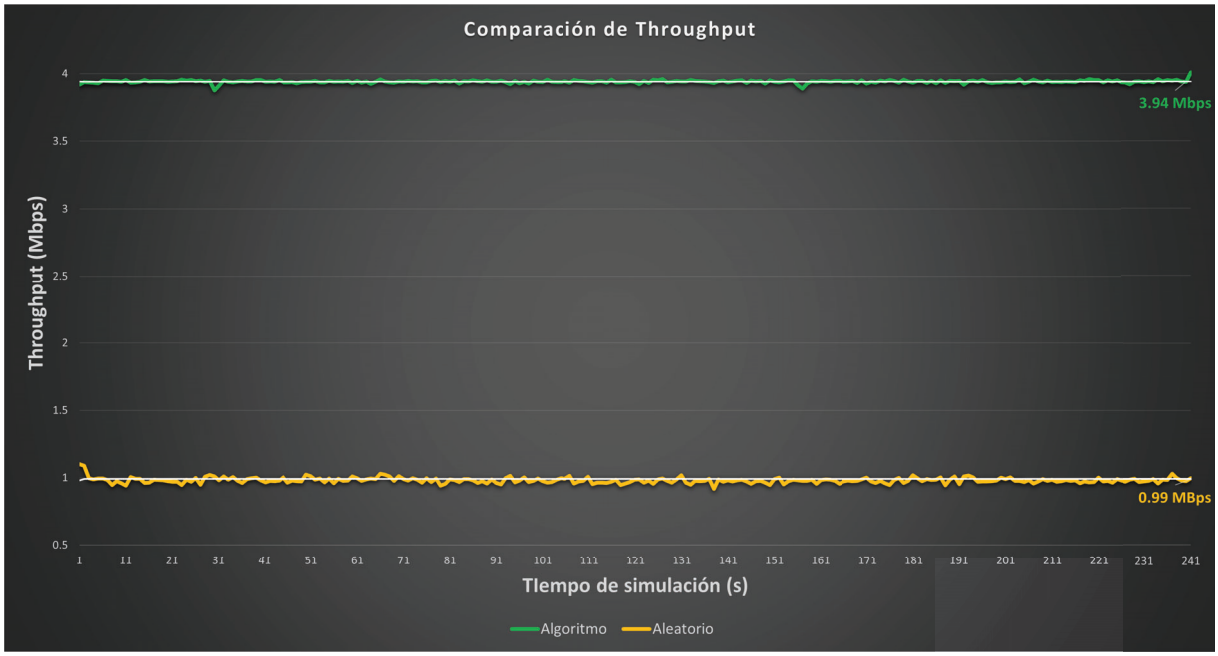


Figura 4.9: Resultados del *Throughput*.

La figura 4.9 muestra el comportamiento del *throughput* para ambos casos. En color amarillo se muestran los resultados correspondientes a la asignación de canales aleatoria. Mientras la línea en color verde corresponde a los resultados obtenidos cuando se aplica la asignación de canales. Las líneas blancas continuas representan los valores promedio para cada escenario. Al comparar el *throughput* del algoritmo propuesto con respecto al obtenido bajo la asignación aleatoria, se puede determinar que existe una mejoría del 397.97 %. Nuevamente esto es una mejora muy significativa en la tasa de transferencia dentro de la red.

Por otro lado, los resultados correspondientes al parámetro de *jitter* se muestran en la figura 4.10. En esta figura se puede observar que si bien existe una mejora del *jitter* entre los casos en que se tienen una asignación aleatoria (lineas azules) comparado cuando la asignación de canales se realiza con ayuda del algoritmo propuesto (lineas verdes), esta diferencia no es muy grande. En este caso la mejora en los tiempos de retraso corresponde a una reducción de $0.68\,\mathrm{ms}$ con respecto a los tiempos promedio (lineas continuas).

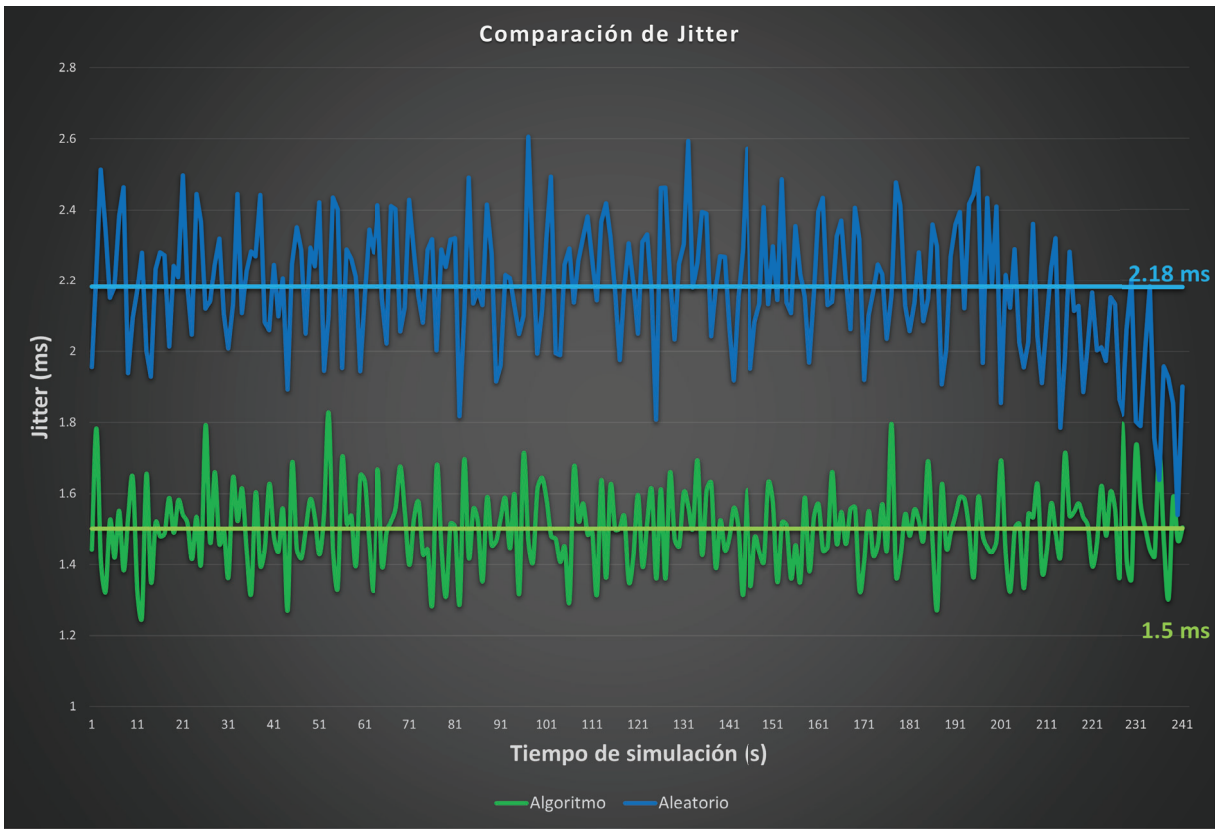


Figura 4.10: Resultados del *Jitter*.

Si bien ambos parámetros se traducen en mejorías en la red con una asignación de canales a través del algoritmo propuesto, en el caso del *jitter*, la mejora es mínima, y de hecho los tiempos obtenidos con una asignación aleatoria para este parámetro son muy parecidos. Esto se debe a que Mininet no tiene la capacidad de replicar fenómenos las multitrayectorias, obstáculos que se pueden presentar en un entorno real que modifican el comportamiento de las señales. Es por esto que sin importar los cambios que se realicen los resultados obtenidos en una simulación tenderán a no tener grandes fluctuaciones en sus valores.

Considerando los resultados obtenidos en las diferentes pruebas, tanto en el escenario físico como en el escenario simulado. Se puede afirmar que el algoritmo propuesto proporciona mejoras considerables en la eficiencia de la red, las cuales se traducen en un aumento significativo en las tasas de transferencia, así como una disminución importante en los tiempos de retraso al momento del envío de paquetes. Por otro lado, el uso de la clasificación por *clusters* como etapa previa a la asignación de canales por medio del algoritmo de coloración permite reducir la complejidad de este último, ya que en el peor de los casos se tendría que considerar toda la red como una vecindad para realizar la asignación de canales. Además, la separación por *clusters* utilizando el índice de Jaccard permite identificar aquellos AP que tienden a generar mayor interferencia entre ellos, al comparar varias métricas como la calidad de la señal, la distancia, la interferencia entre canales, entre otras.

Comparando con de otros trabajos en la literatura, en los cuales se aplica un algoritmo de

coloración a toda la red, en este trabajo de tesis primero se identifican las zonas de la red que presentan un mayor grado de interferencia, para después realizar una asignación de canales en cada una de estas zonas.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

Hoy en día, la asignación de canales en redes inalámbricas es uno de los problemas más importantes al momento de desplegar este tipo de redes. Si bien ya se cuenta con la banda de los 5 GHz, muchas de las redes inalámbricas siguen siendo desplegadas sobre la banda de los 2.4 GHz. El problema de esta banda es que además de WiFi existen otras tecnologías trabajando sobre esta misma frecuencia. Es por esto que la asignación de canales se vuelve una estrategia importante con la cual se pretende aminorar el problema de interferencia que se pueda generar por los propios puntos de acceso de la red y de redes externas que lleguen a encontrarse cerca. En este trabajo se presentó un algoritmo en dos etapas que, además de aprovechar la teoría de coloración de grafos y la división de conjuntos por clusters, hace uso del paradigma de SDN para la gestión y comunicación de la red, y de esta forma puede aplicar el algoritmo propuesto de forma simple, además de tener las bondades que brindan las SDN como son la división en las capas de infraestructura y datos, la flexibilidad para escalar la arquitectura de la red, la visualización global de la red por medio de la capa de control centralizada, entre otras ventajas.

Por otro lado, uno de los puntos principales de este trabajo es realizar la implementación del algoritmo propuesto dentro de un entorno físico o real, con el fin de observar el comportamiento del algoritmo y comprobar que realmente se tenga un efecto positivo sobre una red, aUn y cuando lleguen a existir más redes inalámbricas a su alrededor que pueden causar interferencia con la red de estudio.

Como resultado de las pruebas realizadas en el escenario físico se pudo observar una mejora en los parámetros medidos en la red. En el caso del *throughput* hubo una mejora significativa del $330.7\,\%$ para el escenario en el que se realizó la asignación de canales resultado del algoritmo propuesto, en comparación con el escenario en el que se tenía una asignación aleatoria de los canales. Para el caso del *jitter*, de igual manera se tuvo una mejora significativa, ya que los tiempos de retardo entre los paquetes se redujeron de $13.75\,\mathrm{ms}$ en promedio en un escenario con asignación aleatoria, a $1.27\,\mathrm{ms}$ promedio una vez aplicada la asignación obtenida por el algoritmo.

Finalmente, se realizó la simulación del escenario físico con ayuda de Mininet WiFi con el fin de observar qué tanta diferencia puede existir en el comportamiento de los resultados de las medición de los parámetros *throughput* y *jitter*. Como resultado de realizar las mismas pruebas que en el escenario físico, también se obtuvieron mejoras en ambos parámetros. En el caso del *throughput*, la mejora fue de casi el $400\,\%$, lo cual es una mejora muy significativa, en cuanto a las tasas de transmisión. Para el caso del *jitter* también se obtuvo una mejora, aunque en este caso fue mínima, ya que desde un principio en el escenario con una asignación aleatoria los tiempos promedio de retraso se encontraban al rededor de $2.2\,\mathrm{ms}$; una vez aplicado el algoritmo de asignación de canales, estos tiempos bajaron hasta $1.5\,\mathrm{ms}$ en promedio.

Algo interesante que se puede notar al comparar las gráficas de resultados obtenidas en el escenario real y la simulación son las variaciones de valores tanto en la asignación aleatoria, como en la asignación con ayuda del algoritmo. En el caso de los resultados de la simula-

ción, las variaciones entre los valores resultantes son mínimas, podría decirse que los valores pueden llegar a ser constantes, en ambos parámetros. Sin embargo, en el caso de los valores arrojados en el escenario físico, es claro que que existen variaciones notables en los valores correspondientes a cada parámetro. Esto se debe a la interferencia que sufre la red física, por parte de las redes que se encuentran a su alrededor. Además de los obstáculos o paredes que puede llegar a haber y que generan fenómenos como las multitrayectorias que van afectando el desempeño de la red.

5.2. Verificación de la hipótesis

Retomando la hipótesis que se presentó en la sección 1.2:

La combinación de un algoritmo de clustering y un algoritmo de coloración de grafos para el plano de control de una SDN permite optimizar la asignación de canales en una red WiFi de manera dinámica.

Con base en la Sección 4.3, la hipótesis anterior es verificada, ya que el algoritmo propuesto funciona correctamente y proporciona una optimización considerable en la red. Además, se concretó la implementación de mensajería dentro de una red SDN real para la recolección de información con la cual trabaja el algoritmo propuesto. Esta última parte es una aportación importante, ya que a lo largo del desarrollo de este trabajo de tesis, el estado del arte con respecto a la creación de mensajes e intercambio de información personalizada dentro de una SDN es mínima. Sin embargo, en este trabajo se explica detalladamente cómo se realiza este proceso de comunicación y la creación de los diferentes mensajes con los cuales se realiza el intercambio de información entre el controlador Ryu y los AP de la red.

Si bien el hecho de implementar el algoritmo propuesto en una red física permite visualizar los efector reales de interferencia, si se quisiera probar el desempeño en una red de mayor tamaño, cada vez sería más difícil debido a la cantidad de equipo necesario, así como el tamaño del espacio requerido para su despliegue. Para estos casos el uso de simuladores cobran relevancia. Aunque se pudo observar que no se puede obtener un comportamiento similar a la realidad al momento de desplegar la red en un emulador debido a la dificultad de modelar fenómenos físicos como las múltitrayectorias, las interferencias generadas por redes aledañas o dispositivos de comunicación cercanos que se encuentren trabajando sobre las mismas bandas.

Apéndice A

Instalación de OpenvSwitch

La versión estable de OpenVSwitch utilizada para esta tesis es la 2.13.4.
El proceso de instalación es el siguiente:

- Se instalan las herramientas de **uuid**, en caso de no tenerlas en el sistema, esta herramienta nos ayudará a generar los *id* de la base de datos de OpenVSwitch.

```
sudo apt-get install uuid-runtime
```

- Se descarga la versión 2.13.4 de OpenVSwitch, del sitio oficial.

```
wget https://www.openvswitch.org/releases/openvswitch-2.13.4.tar.gz
```

- Se desempaqueta la carpeta descargada y se ingresa a la misma.

```
tar -xvf openvswitch-2.13.4.tar.gz
cd openvswitch-2.13.4
```

- Una vez dentro, se ingresa como superusuario y se instalan las bibliotecas de Python necesarios para el funcionamiento de OpenVSwitch.

```
sudo su
apt-get install python-simplejson python-qt4 libssl-dev
python-twisted-conch automake autoconf gcc uml-utilities
libtool build-essential pkg-config
```

- Realizada la instalación anterior se verifica la versión del kernel del sistema.

```
uname -r
```

- Una vez conocida la versión del kernel se buscan y se instalan los headers que sean compatibles con la versión o los más actuales.

```
sudo apt-cache search linux-headers
apt-get install -y linux-headers-4.9.0-6-rpi
```

- Ya descargados se realiza la configuración de OpenVSwitch considerando los headers descargados anteriormente.

```
./configure --with-linux=/lib/modules/4.9.0-6-rpi/build
```

- Una vez configurado se realiza la compilación e instalación de OpenVSwitch en el sistema.

```
make && make install
```

- Una vez instalado se entra en la carpeta `datapath/linux` y se añade el modulo `openvswitch`

```
cd datapath/linux
modprobe openvswitch
echo "openvswitch" >> /etc/modules
```

- Se crea la base de datos de OpenVSwitch.

```
mkdir -p /usr/local/etc/openvswitch
ovsdb-tool create /usr/local/etc/openvswitch/conf.db
vswitchd/vswitch.ovsschema
```

- Finalmente, se inicia OpenVSwitch.

```
sudo /usr/local/share/openvswitch/scripts/ovs-ctl --system-id=random start
```

Apéndice B

Instalación de controlador Ryu

La instalación del controlador Ryu también se realizo de forma manual, ya que se modificaron partes del código para la creación de mensajes.

Por lo tanto la instalación se realiza mediante las siguientes instrucciones:

```
$ sudo apt-get install git python-dev python-setuptools python-pip
$ git clone https://github.com/osrg/ryu.git
$ cd ryu
$ sudo pip3 install .
```

Apéndice C

Instalación de Software para la creación de Access Point

Este proceso de instalación se realizo para cada una de las Raspberry Pi que se usaron como Accees Points dentro de la red SDN creada.

C.1. Instalación de Software

Para funcionar como punto de acceso, la Raspberry Pi necesita tener instalado el paquete de software de punto de acceso `hostapd`:

```
$ sudo apt install hostapd
```

Ya instalado se habilita el servicio de punto de acceso inalámbrico y se configura para que comience cuando la Raspberry Pi arranque:

```
$ sudo systemctl unmask hostapd
$ sudo systemctl enable hostapd
```

Con el fin de proporcionar servicios de gestión de red (DNS, DHCP) a los clientes inalámbricos, la Raspberry Pi necesita tener el paquete de software `dnsmasq`:

```
$ sudo apt install dnsmasq
```

Finalmente, se instala **netfilter-persistent** y su plugin **iptables-persistent**. Esta herramienta ayuda al guardar reglas de firewall y restaurarlas cuando la Raspberry Pi arranca:

```
$ sudo DEBIAN_FRONTEND=noninteractive apt install -y netfilter-persistent
iptables-persistent
```

C.2. Configuración del Router de red

Cada Raspberry Pi ejecutará y gestionará una red inalámbrica independiente. También enrutará entre las redes inalámbricas y Ethernet, proporcionando acceso a Internet a los clientes inalámbricos.

C.2.1. Configuración de la interfaz inalámbrica

Raspberry Pi ejecuta un servidor DHCP para la red inalámbrica; esto requiere una configuración IP estática para la interfaz inalámbrica (wlan0) en la Raspberry Pi. El Raspberry Pi también actúa como router en la red inalámbrica, y como es habitual, le daremos la primera dirección IP de la red: 192.168.10.1.

Para configurar la IP estática, se debe de editar el archivo de configuración de `dhcpcd`:

```
$ sudo nano /etc/dhcpcd.conf
```

Al final de este archivo se añade los siguientes parámetros:

```
interface wlan0
    static ip_address=192.168.10.1/24
    nohook wpa_supplicant
```

C.2.2. Habilitando el enrutamiento y enmascaramiento de IP

En esta sección se configuran las Raspberry Pi para permitir que los clientes inalámbricos accedan a computadoras en la red principal (Ethernet) y desde allí a Internet.

Para habilitar el enrutamiento, es decir, para permitir que el tráfico fluya de una red a otra, se crea en las Raspberry Pi, un archivo llamado routed-ap por medio del siguiente comando:

```
$ sudo nano /etc/sysctl.d/routed-ap.conf
```

El contenido de este archivo es el siguiente:

```
# Enable IPv4 routing
net.ipv4.ip_forward=1
```

Al habilitar el enrutamiento , se permite que los hosts conectados a la red 192.168.10.1/24 puedan alcanzar la LAN y el enrutador principal que les dará salida a Internet. Para permitir el tráfico entre clientes en esta red inalámbrica privada e Internet sin cambiar la configuración del router principal,la Raspberry Pi como AP puede sustituir la dirección IP de los clientes inalámbricos con su propia dirección IP en la LAN utilizando una regla de firewall **"masquerade"**, de esta forma:

- El router principal verá todo el tráfico saliente de los clientes inalámbricos como procedente de la Raspberry Pi, lo que permite la comunicación con Internet.
- La Raspberry Pi recibirá todo el tráfico entrante, sustituirá las direcciones IP y reenviará el tráfico al cliente inalámbrico original.

Para configurar este proceso se debe añadir una nueva regla al firewall de cada Raspberry Pi:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Una vez establecida la regla se deben de guardar las reglas actuales para IPv4 (incluyendo la regla anterior) e IPv6 para que se carguen en el arranque por el servicio `netfilter-persistent`:

```
sudo netfilter-persistent save
```

Las reglas de filtrado se guardan en el directorio `/etc/iptables/`.

C.2.3. Configuración de los servicios DHCP y DNS

Los servicios de DHCP y DNS son provistos por `dnsmasq`. Si bien la configuración por default sirve para la creación de una red inalámbrica, solo nos limitaremos a usar algunas de las funciones.

De esta forma crearemos un nuevo archivo de configuración de la siguiente forma:

```
$ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
$ sudo nano /etc/dnsmasq.conf
```

Al nuevo archivo creado añadiremos los siguientes parámetros:

```
interface=wlan0 # Listening interface
dhcp-range=192.168.10.2,192.168.10.20,255.255.255.0,24h
                # Pool of IP addresses served via DHCP
domain=wlan     # Local wireless DNS domain
address=/gw.wlan/192.168.10.1
                # Alias for this router
```

A partir del archivo anterior los AP entregaran direcciones IP entre 192.168.10.2 y 192.168.10.20, con un tiempo de arrendamiento de 24 horas, a clientes DHCP inalámbricos, que se conecten a dicho AP.

C.2.4. Configuración de los parámetros del Access Point

Para este proceso se debe crear el archivo de configuración `hostapd` dentro del directorio `/etc/hostapd/hostapd.conf`, en este archivo es donde se declararan algunos de los parámetros importantes de cada AP.

El archivo creado contendrá las siguientes especificaciones:

```
country_code=MX
interface=wlan0
ssid=AP1
hw_mode=g
channel=1
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=RaspiAP1
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCM
```

De acuerdo a este archivo de configuración el AP asume que usa el canal 1, que el nombre de la red es **AP1** y la contraseña es **RaspiAP1**. La contraseña asignada a la red debe de contener entre 8 y 64 caracteres. Por medio de la linea `country_code=MX`, se configura que los dispositivos conectados trabajen bajo las frecuencias correspondientes al país en cuestión, en este caso México. Enes te caso todos los AP utilizados trabanjan bajo las frecuencias de 2.4 GHz, por lo que el protocolo de operación es g, como se muestra en la linea `hw_mode=g`. Si se optará por trabajar en las frecuencias de 5 GHz se debe de modificar el modo de operación, `hw_mode=a`. Se debe de tomar en cuenta que, al cambiar el modo de operación también debe de cambiarse el canal de acuerdo a lo establecido.

C.2.5. Despliegue de la red inalámbrica

Una vez establecidos todos los parámetros del AP solo queda reiniciar la Raspberry Pi y verificar que el AP ya se encuentre visible y disponible para los dipositivos que deseen conectarse,

estos visualizaran al AP por medio del nombre establecido en `/etc/hostapd/hostapd.conf`, por por medio de la contraseña establecida en el mismo archivo, debe de ser posible establecer la conexión con la red.

```
sudo systemctl reboot
```

Apéndice D

Códigos para la obtención de información

D.1. Código *cmd.py* para la obtención de parámetros del AP.

```
import os
wifi_channel = os.popen('iw  wlan0 info | grep "channel"')
output = wifi_channel.read()
field = str(output)
channel = field.strip().split(" ")[1]
prueba = os.popen('touch /home/pi/PRUEBACMD.txt')

txp = os.popen('iw wlan0 info | grep "txpower"')
output = txp.read()
field = str(output)
tx_power = field.strip().split(" ")[1]

f = open ('/home/pi/openvswitch-2.13.4/ofproto/data.txt','w')
f.write(channel+"\n")
f.write(tx_power)
f.close()
```

D.2. Código *ap_data.py* para el escaneo de vecinos y sus parámetros.

```
import os
from typing import List
import numpy as np
aps = ['AP1','AP2','AP3','AP4','AP5','AP6','AP7','AP8','AP9','AP10','AP11','AP12','AP13','AP14
    ','AP15','AP16']
aps_found =[]
rssi_dict = {}
loq_dict = {}
ch_dict = {}
def get_channel(c):
   channel = c.split(":")
#   print(channel)
   if channel[0]='                    Channel':

       channel = channel[1]
   else:
       channel=0

   return int(str(channel))

def get_quality(q):
```

```
    quality= q.split("=")
    rssi = quality[2].split(" ")[0]
    qualy = quality[1].split("/")[0]
    return int(str(qualy)), int(str(rssi))

def get_ESSID(es):
    print(es)
    essid = str(es.split(":")[1])
    essid = essid.replace('"','')
    return essid


def av_values():
    for essid in aps_found:
        rssi_array = np.array(rssi_dict[essid])
        rssi = int(np.mean(rssi_array))
        loq_array = np.array(loq_dict[essid])
        loq = int(np.mean(loq_array))
        channel = ch_dict[essid]
        AP = [essid, channel,rssi,loq]
        listAP.append(AP)

n=1
while n<15:
    wifi_channel = os.popen('sudo iwlist wlan0 scanning | egrep "Channel|Quality|ESSID"')
    output = wifi_channel.read()
    #print(output)
    field = list(output.split ("\n"))
    field.pop(-1)

    flag = 1
    l =[]
    data =[]
    for i in range(0,int(len(field)/4)):
        l=field[i*4:i*4+4]
        data.append(l)

    listAP = []
    for i in data:
        essid = get_ESSID(i[3])
        if essid in aps:
            if essid not in aps_found:
                aps_found.append(essid)
            channel = get_channel(i[0])
            qualy, rssi = get_quality(i[2])
            if essid in rssi_dict:
                rssi_dict[essid].append(rssi)
                loq_dict[essid].append(qualy)
            else:
                rssi_dict[essid] = [rssi]
                loq_dict[essid] = [qualy]

            if essid not in ch_dict:
                ch_dict[essid] = channel

    n=n+1


av_values()

for i in listAP:
    print(i)

f = open ('/home/pi/openvswitch-2.13.4/ofproto/neighbours.txt','w')
f.write(str(listAP)+"\n")
f.close()
```

D.3. Código *change_channel.py* para realizar el cambio de canal en el AP.

```
import os
import sys
import time

prueba = os.popen('touch /home/pi/EXITO2.txt')


f=open('/etc/hostapd/hostapd.conf','r')
data = f.readlines()
ch=sys.argv[1]
data[4]='channel='+ch+'\n'
f=open('/etc/hostapd/hostapd.conf','w')
f.writelines(data)

restart = os.popen('sudo systemctl restart hostapd.service')
time.sleep(5)
```

Apéndice E

Código del algoritmo de asignación de canales.

```
from select import select
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from scipy import special as sp
import math
from collections import OrderedDict
import random
from itertools import combinations as comb
import matplotlib.pyplot as plt
from accessPoint import *
from GraphS import *

class Algoritmo():

    def __init__(self):
        self.APS = []
        self.nAP = None
        self.ACN=None #Matriz de adyacencias
        self.ACJ=None #Matriz de Jaccard
        self.Clusters = []
        self.AC_sim =[]
        self.AP_grado ={}
        self.P_tx=15

    def configuracion(self,Coord,channels):
        self.APS,self.ACN = ConjuntoAP(Coord,channels)
        self.nAP = len(self.APS)
        self.ACJ = np.zeros((self.nAP,self.nAP))
        self.ajuste()


    ###Proceso de lógica difusa###
    def calculo_interferencia(self, loq,dist):

        x_link_of_q = np.arange(10, 100+0.001, 0.1)
        x_distance = np.arange(0, 30+0.001, 0.1)
        x_interference = np.arange(0, 10+0.001, 0.1)

        link_of_qual = ctrl.Antecedent(x_link_of_q, 'link_of_qual')
        distance = ctrl.Antecedent(x_distance, 'distance')
        interference = ctrl.Consequent(x_interference, 'interference')



        link_of_qual['poor'] = fuzz.trimf(link_of_qual.universe,[10,10,55])
        link_of_qual['avg'] = fuzz.trimf(link_of_qual.universe,[10,55,100])
        link_of_qual['good'] = fuzz.trimf(link_of_qual.universe,[55,100,100])
```

```
        distance['close'] = fuzz.trimf(distance.universe, [0, 0, 15] )
        distance['medium'] = fuzz.trimf(distance.universe, [0, 15, 30])
        distance['far'] = fuzz.trimf(distance.universe, [15, 30, 30])

        interference['low'] = fuzz.trimf(interference.universe, [0, 0, 5])
        interference['medium'] = fuzz.trimf(interference.universe, [0, 5, 10])
        interference['high'] = fuzz.trimf(interference.universe, [5, 10, 10])




        rule1 = ctrl.Rule(link_of_qual['poor'] & distance['close'], interference['medium'])
        rule2 = ctrl.Rule(link_of_qual['poor'] & distance['medium'], interference['medium'])
        rule3 = ctrl.Rule(link_of_qual['poor'] & distance['far'], interference['low'])
        rule4 = ctrl.Rule(link_of_qual['avg'] & distance['close'], interference['high'])
        rule5 = ctrl.Rule(link_of_qual['avg'] & distance['medium'], interference['low'])
        rule6 = ctrl.Rule(link_of_qual['avg'] & distance['far'], interference['low'])
        rule7 = ctrl.Rule(link_of_qual['good'] & distance['close'], interference['high'])
        rule8 = ctrl.Rule(link_of_qual['good'] & distance['medium'], interference['high'])
        rule9 = ctrl.Rule(link_of_qual['good'] & distance['far'], interference['medium'])


        interference_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
    rule7, rule8, rule9])
        interference_level = ctrl.ControlSystemSimulation(interference_ctrl)

        interference_level.input['link_of_qual'] = loq
        interference_level.input['distance'] = dist
        # Crunch the numbers
        interference_level.compute()

        interferencia = interference_level.output['interference']

        return interferencia


    def jaccard_similarity(self,x,y):

        intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
        union_cardinality = len(set.union(*[set(x), set(y)]))

        return intersection_cardinality/float(union_cardinality)

    def calculo_distancia(self):
        return np.random.uniform(0,15,1)

    def calculo_link_q(self):
        return np.random.random()

    def qfunc(self,x):
        return 0.5 - 0.5*sp.erf(x/math.sqrt(2))

    def rssi_level(self,rssi):
        print(rssi)
        if rssi <=-10 and rssi >=-59:
            level= 'low'
        elif rssi <=-60 and rssi >=-79:
            level = 'medium'
        elif rssi <=-80:
            level = 'high'
        return level

    def proba_enlace(self,P_tx,dist):
        P_min = -90
        d0 = 1
        g=5.1
        sig=12.9
        l = (3*math.pow(10,8))/(2.4*math.pow(10,9))
        K = 20*math.log10(l/(4*math.pi*d0))
        F = (P_min-(P_tx+K-10*g*math.log10(dist/d0)))/sig
```

```
        Q = self.qfunc(F)
        return Q

    def busqueda_AP(self,AP):
        for i in self.APS:
            if AP == i.name:
                return i

    def busqueda_vecinos(self,vecinos,AP):
        for i in vecinos:
            if AP == i[0]:
                return i

    def ajuste(self):

        for i in range(0,self.nAP):
            g = len(self.APS[i].neighbours)
            self.AP_grado[self.APS[i].name] = g

            for j in self.APS[i].neighbours:
                vecino = j
                v = self.busqueda_AP(vecino[0])
                no_v = int(v.name[2:])


                ap1 = self.busqueda_vecinos(v.neighbours,self.APS[i].name)

                if ap1 != None:
                    #Valores para Link_quality
                    r1 = int(vecino[3]) #Valor observado por el AP1 para el AP2
                    r2 = int(ap1[3]) #Valor observado por el AP2 para el ap1
                    loq1 = (r1/70)*100
                    loq2 = (r2/70)*100
                    print(loq1,loq2)
                    link_q1 = round(self.proba_enlace(self.APS[i].P_tx ,vecino[4]),2)
                    link_q2 = round(self.proba_enlace(v.P_tx,ap1[4]),2)
                    rssi_1 = self.rssi_level(vecino[2])
                    rssi_2 = self.rssi_level(ap1[2])

                    interf1 = round(self.calculo_interferencia(loq1 ,vecino[4])) #calculo de
la interferencia generada por el primer AC
                    interf2 = round(self.calculo_interferencia(loq2,ap1[4])) #calculo de la
interferencia generada por el segundo AC
                    AC1=[self.APS[i].channel,interf1,link_q1,rssi_1] #[canal,interferencia,
link q]
                    AC2=[vecino[1],interf2,link_q2, rssi_2] #[canal,interferencia,link q]
                    print(self.APS[i].name,AC1)
                    print(v.name,AC2)
                    print('\n')
                    jacc_s =round(self.jaccard_similarity(AC1,AC2) ,2)
                    self.AC_sim.append([i,no_v,jacc_s])
                    self.ACJ[i][no_v-1]=jacc_s
        print(self.AP_grado)

    ### Proceso de clustering con el indice de Jaccard ###
    def clustering(self,jth):
        Jacc =[]
        j1 =[]
        cluster=[]
        for i in range(0,len(self.ACN)):
            cluster.append(i+1)
            for j in range(0,len(self.ACN)):
                if self.ACN[i][j]==1:
                    if self.ACJ[i][j]<=jth:
                            cluster.append(j+1)
                            j1.append(self.ACJ[i][j])
            if cluster !=[]:
                self.Clusters.append(cluster)
                Jacc.append(j1)


```

```
            cluster=[]
188         j1 = []
        print('Custer', self.Clusters)
190     k=1
        self.Clusters.sort(key=len)
192     for i in self.Clusters:
            for elem in i:
194             for  j in range(k,len(self.Clusters)):
                    if elem in self.Clusters[j]:
196                     self.Clusters[j].remove(elem)
            k=k+1
198     while (self.Clusters.count([])):
            self.Clusters.remove([])
200
        print('Custersss', self.Clusters)
202

204 def coloring(self):
        for c in self.Clusters:
206         wp_color={}
            g= Graph()
208         if len(c) ==1:
                g.add_edge('AP'+str(c[0]),'AP'+str(c[0]),1)
210             dict_wp = g.wp_coloring()
                self.change_ch(dict_wp)
212         else:

214             for ap in c:
                    for p in c:
216                     if self.ACN[ap-1][p-1] == 1:
                            g.add_edge('AP'+str(ap),'AP'+str(p),1)
218             dict_wp =  g.wp_coloring()
                self.change_ch(dict_wp)
220

222 def change_ch(self,dict_wp):
        ch = {0:1,1:6, 2:11, 3:4, 4:9, 5:13} #llave : canal
224     for i in dict_wp:
            for j in self.APS:
226             if j.name == i:
                    channel = ch[dict_wp[i]]
228                 j.channel = channel
                    break
230


232


234 def graficas_color(self):
        color = {1:'red',6:'blue',11:'green', 4:'yellow',9:'purple',13:'cyan'}
236     f1=plt.subplot()
        ax = plt.gca()
238     ax.set_title('Coloración de Canales')
        for ap  in self.APS:
240
            x = ap.coordX
242         y = ap.coordY
            col= color[ap.channel]
244         ax.scatter(x,y,color=str(col))
            ax.annotate(ap.name,(x,y),ha='left')
246         ax.annotate(ap.channel,(x,y),ha='right')

248     plt.show()

250


252 def graficas(self):
        color = ['blue', 'green','yellow','orange','purple','red','cyan','magenta','olive', '
    gray','brown','lime']
254     ax = plt.gca()
        c=0
256     for i in self.Clusters:
```

```
            col= color[c]
            for j in i:
                x = self.APS[j-1].coordX
                y = self.APS[j-1].coordY
                ax.scatter(x,y,color=str(col))
                ax.annotate(self.APS[j-1].name,(x,y),ha='left')
                ax.annotate(self.APS[j-1].channel,(x,y),ha='right')

            c=c+1
        plt.show()


    def pruebas(self):
        jth=0.1
        cn = []
        tamano_cluster = []
        while jth < 1:
            print('JACCARD: ', jth)
            self.clustering(jth)
            self.graficas()

            for c in self.Clusters:
                cn.append(len(c))

            cn = np.array(cn)
            cn_mean = np.mean(cn)
            tamano_cluster.append(cn_mean)
            cn = []
            self.Clusters = []
            jth = jth+0.1
        print(tamano_cluster)
```

Apéndice F

Códigos del controlador Ryu para el intercambio de información

F.1. Programa para la creación de parámetros de AP

```
import os
from typing import List
import numpy as np
aps = ['AP1','AP2','AP3','AP4','AP5','AP6','AP7','AP8','AP9','AP10','AP11','AP12','AP13','AP14
    ','AP15','AP16']
aps_found =[]
rssi_dict = {}
loq_dict = {}
ch_dict = {}
def get_channel(c):
   channel = c.split(":")
#   print(channel)
   if channel[0]=='                    Channel':

       channel = channel[1]
   else:
       channel=0

   return int(str(channel))

def get_quality(q):
    quality= q.split("=")
    rssi = quality[2].split(" ")[0]
    qualy = quality[1].split("/")[0]
    return int(str(qualy)), int(str(rssi))

def get_ESSID(es):
    print(es)
    essid = str(es.split(":")[1])
    essid = essid.replace('"','')
    return essid


def av_values():
    for essid in aps_found:
        rssi_array = np.array(rssi_dict[essid])
        rssi = int(np.mean(rssi_array))
        loq_array = np.array(loq_dict[essid])
        loq = int(np.mean(loq_array))
        channel = ch_dict[essid]
        AP = [essid, channel,rssi,loq]
        listAP.append(AP)

n=1
while n<15:
```

```
    wifi_channel = os.popen('sudo iwlist wlan0 scanning | egrep "Channel|Quality|ESSID"')
    output = wifi_channel.read()
    #print(output)
    field = list(output.split ("\n"))
    field.pop(-1)

    flag = 1
    l =[]
    data =[]
    for i in range(0,int(len(field)/4)):
        l=field[i*4:i*4+4]
        data.append(l)

    listAP = []
    for i in data:
        essid = get_ESSID(i[3])
        if essid in aps:
            if essid not in aps_found:
                aps_found.append(essid)
            channel = get_channel(i[0])
            qualy, rssi = get_quality(i[2])
            if essid in rssi_dict:
                rssi_dict[essid].append(rssi)
                loq_dict[essid].append(qualy)
            else:
                rssi_dict[essid] = [rssi]
                loq_dict[essid] = [qualy]

            if essid not in ch_dict:
                ch_dict[essid] = channel

    n=n+1


av_values()

for i in listAP:
    print(i)

f = open ('/home/pi/openvswitch-2.13.4/ofproto/neighbours.txt','w')
f.write(str(listAP)+"\n")
f.close()
```

F.2. Programa de ejecución del algoritmo de asignación

En este código ya se incluye la información necesaria para ejecutar el algoritmo sin necesidad de recolectar la información

```
from mapa import *
from accessPoint import *
from Algoritmo import *
Coord = [(13, 34), (0, 0), (26, 45), (24, 20), (29, 25), (24, 27), (23, 34), (27, 55), (34,
    47), (7, 15), (12, 25), (6, 45), (0, 41), (28, 35), (4, 35), (5, 5)]
print('PUNTOS',Coord)

info = [['AP1', 1, 31.0, [['AP7', 1, -62, 47], ['AP10', 1, -77, 32], ['AP11', 6, -67, 42], ['
    AP6', 11, -66, 43], ['AP3', 11, -82, 27], ['AP15', 11, -81, 28], ['AP9', 11, -80, 29], ['
    AP4', 1, -86, 23], ['AP14', 6, -85, 24], ['AP12', 11, -76, 33], ['AP13', 1, -81, 28], ['
    AP5', 6, -67, 33]]],['AP2', 6, 31.0, [['AP16', 1, -65, 44], ['AP10', 1, -80, 29], ['AP1',
    1, -87, 22], ['AP11', 6, -84, 25], ['AP5', 6, -85, 24], ['AP12', 11, -78, 31], ['AP14', 6,
     -86, 23]]],['AP3',11,31,[['AP7', 1, -75, 34], ['AP1', 1, -83, 26], ['AP4', 1, -82, 27], [
    'AP12', 11, -82, 27], ['AP14', 6, -62, 47], ['AP8', 6, -64, 45], ['AP5', 6, -78, 32], ['
    AP11', 6, -82, 27], ['AP15', 11, -91, 18], ['AP9', 11, -72, 38], ['AP6', 11, -76, 33], ['
    AP16', 1, -89, 21]]],['AP4', 1, 31.0, [['AP5', 6, -90, 20],['AP6', 11, -14, 70]]],['AP5'
    ,6,31,[['AP4', 1, -55, 54], ['AP7', 1, -77, 32], ['AP14', 6, -64, 45], ['AP8', 6, -75,
    34], ['AP6', 11, -62, 47], ['AP9', 11, -72, 37], ['AP3', 11, -75, 34], ['AP11', 6, -74,
    35], ['AP12', 11, -83, 27], ['AP15', 11, -83, 27], ['AP1', 1, -84, 25]]],['AP6', 11,
    31.0,[['AP1', 1, -68, 41], ['AP3', 11, -80, 29], ['AP9', 11, -81, 28], ['AP5', 6, -64,
```

```
    45], ['AP14', 6, -72, 37], ['AP7', 1, -73, 36], ['AP12', 11, -79, 31], ['AP15', 11, -85,
    24], ['AP2', 6, -86, 24], ['AP13', 1, -86, 24], ['AP4', 1, -78, 32], ['AP11', 6,
    -82,47]]],['AP7',1,31,[['AP13', 1, -75, 35], ['AP4', 1, -78, 32], ['AP11', 6, -70, 40], ['
    AP14', 6, -54, 56], ['AP5', 6, -78, 32], ['AP8', 6, -70, 40], ['AP10', 1, -80, 30], ['AP12
    ', 11, -77, 33], ['AP6', 11, -67, 43], ['AP9', 11, -65, 45], ['AP3', 11, -69, 41]]],['AP8'
    , 6, 31.0, [['AP3', 11, -67, 42], ['AP14', 6, -69, 40], ['AP9', 11, -72, 37], ['AP4', 1,
    -72, 38], ['AP5', 6, -73, 36], ['AP12', 11, -72, 37]]],['AP9',11,31,[['AP14', 6, -53, 56],
     ['AP8', 6, -69, 40], ['AP6', 11, -71, 38], ['AP3', 11, -64, 46], ['AP11', 6, -69, 41], ['
    AP7', 1, -72, 38], ['AP5', 6, -71, 39], ['AP12', 11, -67, 43]]],['AP10',1,31,[['AP16', 1,
    -57, 52], ['AP4', 1, -76, 33], ['AP11', 6, -74, 35], ['AP12', 11, -75, 34], ['AP15', 11,
    -76, 33], ['AP7', 1, -78, 32], ['AP2', 11, -78, 31], ['AP6', 11, -77, 32], ['AP1', 1, -84,
     26], ['AP8', 6, -80, 29]]],['AP11', 6, 31.0, [['AP10', 1, -74, 35], ['AP1', 1, -68, 41],
    ['AP4', 1, -73, 36], ['AP12', 11, -71, 38], ['AP6', 11, -76, 33], ['AP5', 6, -75, 34]]],['
    AP12', 11, 31.0, [['AP13',1,-65,28],['AP16', 1, -45,-44]]],['AP13', 1, 31.0, [['AP12', 11,
     -60,40],['AP15', 11,-78,30]]],['AP14', 6, 31.0, [['AP5', 6, -67, 42], ['AP8', 6, -67,
    42], ['AP2', 6, -85, 24], ['AP9', 11, -59, 50], ['AP3', 11, -73, 36], ['AP11', 6, -85,
    24], ['AP7', 1, -75, 34], ['AP6', 11, -73, 36], ['AP1', 1, -83, 26], ['AP12', 11, -82,
    27], ['AP15', 11, -86, 23], ['AP10', 1, -83]]],['AP15', 11, 31.0, [['AP7', 1, -76, 33], ['
    AP10', 1, -83, 26], ['AP1', 1, -77, 32], ['AP3', 11, -90, 19], ['AP9', 11, -80, 29], ['
    AP12', 11, -64, 45], ['AP13', 1, -82, 27], ['AP6', 11, -89, 20], ['AP14', 6, -87, 23], ['
    AP11', 6, -90, 20], ['AP5', 6, -92, 18], ['AP8', 6, -89,20]]],['AP16', 1, 31.0, [['AP4',
    1, -80, 29], ['AP10', 1, -86, 23], ['AP2', 6, -75, 34], ['AP1', 1, -87, 22], ['AP6', 11,
    -85, 24], ['AP9', 11, -85, 24], ['AP8', 6, -84, 25]]]]

algoritmo = Algoritmo()
algoritmo.configuracion(info,Coord)
print('\n')
mapa()
algoritmo.graficas_color()
algoritmo.clustering(0.6)
mapa()
algoritmo.graficas()

algoritmo.coloring()
mapa()
algoritmo.graficas_color()
```

F.3. Programa de controlador Ryu para la recolección de información

```
from operator import attrgetter
from datetime import datetime
from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


info=[]
i=1
class SimpleMonitor13(simple_switch_13.SimpleSwitch13):
    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)


    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self.logger.info('DATAPATH= %d ', dp.id)
                self.send_desc_stats_request(dp)
            hub.sleep(3)


```

```
    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]


    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)

    def send_desc_stats_request(self, datapath):
        self.logger.debug('REQUEST PARAMS FOR: %016x', datapath.id)
        ofp_parser = datapath.ofproto_parser

        req = ofp_parser.OFPDescStatsRequest(datapath, 0,1)
        datapath.send_msg(req)


    @set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
    def desc_stats_reply_handler(self, ev):
        body = ev.msg.body
        self.logger.debug('DescStats: Name AP = %s Type switch = %s WiFi channel = %s '
                          ' Tx_power = %s Neighbours = %s',
                          body.name_AP,int(body.type_switch), int(body.channel),
                          float(body.tx_power), body.neighbours)
        if int(body.type_switch) == 1:
            name_AP =body.name_AP.decode('utf-8')
            neighbours = body.neighbours.decode('utf-8')
            info.append([name_AP,int(body.channel),float(body.tx_power),neighbours])
            print(info)
```

F.4. Algoritmo de asignación de canales WiFi

```
from select import select
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from scipy import special as sp
import math
from collections import OrderedDict
import random
from itertools import combinations as comb
import matplotlib.pyplot as plt
from accessPoint import *
from GraphS import *

class Algoritmo():

    def __init__(self):
        self.APS = []
```

```
        self.nAP = None
        self.ACN=None #Matriz de adyacencias
        self.ACJ=None #Matriz de Jaccard
        self.Clusters = []
        self.AC_sim =[]
        self.AP_grado ={}
        self.P_tx=15

    def configuracion(self,Coord,channels):
        self.APS,self.ACN = ConjuntoAP(Coord,channels)
        self.nAP = len(self.APS)
        self.ACJ = np.zeros((self.nAP,self.nAP))
        self.ajuste()


    ###Proceso de lógica difusa###
    def calculo_interferencia(self, loq,dist):

        x_link_of_q = np.arange(10, 100+0.001, 0.1)
        x_distance = np.arange(0, 30+0.001, 0.1)
        x_interference = np.arange(0, 10+0.001, 0.1)

        link_of_qual = ctrl.Antecedent(x_link_of_q, 'link_of_qual')
        distance = ctrl.Antecedent(x_distance, 'distance')
        interference = ctrl.Consequent(x_interference, 'interference')



        link_of_qual['poor'] = fuzz.trimf(link_of_qual.universe,[10,10,55])
        link_of_qual['avg'] = fuzz.trimf(link_of_qual.universe,[10,55,100])
        link_of_qual['good'] = fuzz.trimf(link_of_qual.universe,[55,100,100])

        distance['close'] = fuzz.trimf(distance.universe, [0, 0, 15] )
        distance['medium'] = fuzz.trimf(distance.universe, [0, 15, 30])
        distance['far'] = fuzz.trimf(distance.universe, [15, 30, 30])

        interference['low'] = fuzz.trimf(interference.universe, [0, 0, 5])
        interference['medium'] = fuzz.trimf(interference.universe, [0, 5, 10])
        interference['high'] = fuzz.trimf(interference.universe, [5, 10, 10])



        rule1 = ctrl.Rule(link_of_qual['poor'] & distance['close'], interference['medium'])
        rule2 = ctrl.Rule(link_of_qual['poor'] & distance['medium'], interference['medium'])
        rule3 = ctrl.Rule(link_of_qual['poor'] & distance['far'], interference['low'])
        rule4 = ctrl.Rule(link_of_qual['avg'] & distance['close'], interference['high'])
        rule5 = ctrl.Rule(link_of_qual['avg'] & distance['medium'], interference['low'])
        rule6 = ctrl.Rule(link_of_qual['avg'] & distance['far'], interference['low'])
        rule7 = ctrl.Rule(link_of_qual['good'] & distance['close'], interference['high'])
        rule8 = ctrl.Rule(link_of_qual['good'] & distance['medium'], interference['high'])
        rule9 = ctrl.Rule(link_of_qual['good'] & distance['far'], interference['medium'])

        interference_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
    rule7, rule8, rule9])
        interference_level = ctrl.ControlSystemSimulation(interference_ctrl)

        interference_level.input['link_of_qual'] = loq
        interference_level.input['distance'] = dist
        # Crunch the numbers
        interference_level.compute()

        interferencia = interference_level.output['interference']

        return interferencia


    def jaccard_similarity(self,x,y):

        intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
```

```
        union_cardinality = len(set.union(*[set(x), set(y)]))
89
        return intersection_cardinality/float(union_cardinality)
91
    def calculo_distancia(self):
93      return np.random.uniform(0,15,1)

95  def calculo_link_q(self):
        return np.random.random()
97
    def qfunc(self,x):
99      return 0.5 - 0.5*sp.erf(x/math.sqrt(2))

101 def rssi_level(self,rssi):
        print(rssi)
103     if rssi <=-10 and rssi >=-59:
            level= 'low'
105     elif rssi <=-60 and rssi >=-79:
            level = 'medium'
107     elif rssi <=-80:
            level = 'high'
109     return level

111 def proba_enlace(self,P_tx,dist):
        P_min = -90
113     d0 = 1
        g=5.1
115     sig=12.9
        l = (3*math.pow(10,8))/(2.4*math.pow(10,9))
117     K = 20*math.log10(l/(4*math.pi*d0))
        F = (P_min-(P_tx+K-10*g*math.log10(dist/d0)))/sig
119     Q = self.qfunc(F)
        return Q
121
    def busqueda_AP(self,AP):
123     for i in self.APS:
            if AP == i.name:
125             return i

127 def busqueda_vecinos(self,vecinos,AP):
        for i in vecinos:
129         if AP == i[0]:
                return i
131
    def ajuste(self):
133
        for i in range(0,self.nAP):
135         g = len(self.APS[i].neighbours)
            self.AP_grado[self.APS[i].name] = g
137
            for j in self.APS[i].neighbours:
139             vecino = j
                v = self.busqueda_AP(vecino[0])
141             no_v = int(v.name[2:])


143
                ap1 = self.busqueda_vecinos(v.neighbours,self.APS[i].name)
145
                if ap1 != None:
147                 #Valores para Link_quality
                    r1 = int(vecino[3]) #Valor observado por el AP1 para el AP2
149                 r2 = int(ap1[3]) #Valor observado por el AP2 para el ap1
                    loq1 = (r1/70)*100
151                 loq2 = (r2/70)*100
                    print(loq1,loq2)
153                 link_q1 = round(self.proba_enlace(self.APS[i].P_tx ,vecino[4]),2)
                    link_q2 = round(self.proba_enlace(v.P_tx,ap1[4]),2)
155                 rssi_1 = self.rssi_level(vecino[2])
                    rssi_2 = self.rssi_level(ap1[2])
157
```

```
                    interf1 = round(self.calculo_interferencia(loq1 ,vecino[4])) #calculo de
    la interferencia generada por el primer AC
159                 interf2 = round(self.calculo_interferencia(loq2,ap1[4])) #calculo de la
    interferencia generada por el segundo AC
                    AC1=[self.APS[i].channel,interf1,link_q1,rssi_1] #[canal,interferencia,
    link q]
161                 AC2=[vecino[1],interf2,link_q2, rssi_2] #[canal,interferencia,link q]
                    print(self.APS[i].name,AC1)
163                 print(v.name,AC2)
                    print('\n')
165                 jacc_s =round(self.jaccard_similarity(AC1,AC2) ,2)
                    self.AC_sim.append([i,no_v,jacc_s])
167                 self.ACJ[i][no_v-1]=jacc_s
        print(self.AP_grado)
169
    ### Proceso de clustering con el indice de Jaccard ###
171 def clustering(self,jth):
        Jacc =[]
173     j1 =[]
        cluster=[]
175     for i in range(0,len(self.ACN)):
            cluster.append(i+1)
177         for j in range(0,len(self.ACN)):
                if self.ACN[i][j]==1:
179                 if self.ACJ[i][j]<=jth:
                            cluster.append(j+1)
181                         j1.append(self.ACJ[i][j])
            if cluster !=[]:
183             self.Clusters.append(cluster)
                Jacc.append(j1)
185

187         cluster=[]
            j1 = []
189     print('Custer', self.Clusters)
        k=1
191     self.Clusters.sort(key=len)
        for i in self.Clusters:
193         for elem in i:
                for  j in range(k,len(self.Clusters)):
195                 if elem in self.Clusters[j]:
                        self.Clusters[j].remove(elem)
197         k=k+1
        while (self.Clusters.count([])):
199         self.Clusters.remove([])

201     print('Custersss', self.Clusters)

203
    def coloring(self):
205     for c in self.Clusters:
            wp_color={}
207         g= Graph()
            if len(c) ==1:
209             g.add_edge('AP'+str(c[0]),'AP'+str(c[0]),1)
                dict_wp = g.wp_coloring()
211             self.change_ch(dict_wp)
            else:
213
                for ap in c:
215                 for p in c:
                        if self.ACN[ap-1][p-1] == 1:
217                         g.add_edge('AP'+str(ap),'AP'+str(p),1)
                dict_wp =  g.wp_coloring()
219             self.change_ch(dict_wp)

221
    def change_ch(self,dict_wp):
223     ch = {0:1,1:6, 2:11, 3:4, 4:9, 5:13} #llave : canal
        for i in dict_wp:
225         for j in self.APS:
```

```
                if j.name == i:
227                 channel = ch[dict_wp[i]]
                    j.channel = channel
229                 break

231


233
    def graficas_color(self):
235     color = {1:'red',6:'blue',11:'green', 4:'yellow',9:'purple',13:'cyan'}
        f1=plt.subplot()
237     ax = plt.gca()
        ax.set_title('Coloración de Canales')
239     for ap  in self.APS:

241         x = ap.coordX
            y = ap.coordY
243         col= color[ap.channel]
            ax.scatter(x,y,color=str(col))
245         ax.annotate(ap.name,(x,y),ha='left')
            ax.annotate(ap.channel,(x,y),ha='right')
247
        plt.show()
249


251
    def graficas(self):
253     color = ['blue', 'green','yellow','orange','purple','red','cyan','magenta','olive', '
    gray','brown','lime']
        ax = plt.gca()
255     c=0
        for i in self.Clusters:
257         col= color[c]
            for j in i:
259             x = self.APS[j-1].coordX
                y = self.APS[j-1].coordY
261             ax.scatter(x,y,color=str(col))
                ax.annotate(self.APS[j-1].name,(x,y),ha='left')
263             ax.annotate(self.APS[j-1].channel,(x,y),ha='right')

265         c=c+1
        plt.show()
267


269 def pruebas(self):
        jth=0.1
271     cn = []
        tamano_cluster = []
273     while jth < 1:
            print('JACCARD: ', jth)
275         self.clustering(jth)
            self.graficas()
277
            for c in self.Clusters:
279             cn.append(len(c))

281         cn = np.array(cn)
            cn_mean = np.mean(cn)
283         tamano_cluster.append(cn_mean)
            cn = []
285         self.Clusters = []
            jth = jth+0.1
287     print(tamano_cluster)
```

F.5. Programa para el envío de nuevos canales desde el controlador Ryu

```
from operator import attrgetter
from datetime import datetime
from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER, HANDSHAKE_DISPATCHER,
    CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


info=[]
i=1

channels={}

class SimpleMonitor13(simple_switch_13.SimpleSwitch13):
    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)


    def _monitor(self):
        f = open("nuevos_canales.txt")
        channels=f.read()
        f.close()
        while True:
            for dp in self.datapaths.values():
                self.logger.info('DATAPATH= %d ', dp.id)
                ofproto = dp.ofproto
                self.logger.info('BUFFER_ID= %d ',ofproto.OFP_NO_BUFFER)

                if channels[dp.id] in channels:
                    data = channels[dp.id]
                    data = data.encode('utf-8')
                    self.send_echo_request(dp, data)

            hub.sleep(3)



    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]


    def send_echo_request(self, datapath, data):
        ofp_parser = datapath.ofproto_parser

        self.logger.info('ECHO= %d ', datapath.id)
        req = ofp_parser.OFPEchoRequest(datapath, data)
        datapath.send_msg(req)


    @set_ev_cls(ofp_event.EventOFPEchoRequest,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
    def echo_request_handler(self, ev):
        self.logger.debug('OFPEchoRequest received: data=%s',
                          utils.hex_array(ev.msg.data))
```

Bibliografía

[1] M. Rethfeldt, P. Danielis, B. Konieczek, F. Uster, and D. Timmermann, "Integration of qos parameters from ieee 802.11s wlan mesh networks into logical p2p overlays," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pp. 1170–1177, 2015.

[2] M. Kocaleva, A. Stojanova, D. Bikov, T. Balabanov, G. Kobeaga, T. Koca, and T. Ashley, "Self-organized networks," 05 2017.

[3] T. O. N. F. (ONF), "Software-defined networking (sdn) definition." https://opennetworking.org/sdn-definition/, 2022.

[4] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020.

[5] B. P. A. Nygren, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. T. D. Malek, and D. Erickson, "Openflow switch specification version 1.5. 1." https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf, 2015.

[6] V. Khatri, *M.Sc. Thesis: Analysis of OpenFlow Protocol in Local Area Networks*. PhD thesis, 08 2013.

[7] U. of Oregon, "Ryu openflow controller." https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf, 2015.

[8] I. The MathWorks, "Fuzzy inference process." https://www.mathworks.com/help/fuzzy/fuzzy-inference-process.html#FP350, 2022.

[9] G. for geeks., "Comparison between mamdani and sugeno fuzzy inference system." https://www.geeksforgeeks.org/comparison-between-mamdani-and-sugeno-fuzzy-inference-system/, 2020.

[10] A. Mishra, S. Banerjee, and W. Arbaugh, "Weighted coloring based channel assignment for wlans," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, p. 19–31, jul 2005.

[11] M. Seyedebrahimi, F. Bouhafs, A. Raschellà, M. Mackay, and Q. Shi, "Sdn-based channel assignment algorithm for interference management in dense wi-fi networks," in *2016 European Conference on Networks and Communications (EuCNC)*, pp. 128–132, 2016.

[12] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.

[13] M. Abdollahi, W. Ni, M. Abolhasan, and S. Li, "Software-defined networking-based adaptive routing for multi-hop multi-frequency wireless mesh," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13073–13086, 2021.

[14] T. Mahboob, H. Y. Lee, M. Shin, and M. Y. Chung, "Sdn-based centralized channel assignment scheme using clustering in dense wlan environments," *Wireless Personal Communications*, vol. 114, pp. 2693–2716, 10 2020.

[15] L. P. Cevallos Sánchez, "Implementación de redes definidas por software (sdn) sobre redes ieee 802.11 mediante mininet wi-fi.," *Escuela Superior Politécnica de Chimborazo*, 2018.

[16] M. P. Contributors., "Mninnet." http://mininet.org, 2022.

[17] I. S. Association., "Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications." https://standards.ieee.org/ieee/802.11/5536/, 2022.

[18] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), p. 74–87, Association for Computing Machinery, 2018.

[19] ONF, "Sdn architecture overview." https://opennetworking.org/wp-content/uploads/2013/02/SDN-architecture-overview-1.0.pdf, 2014.

[20] O. N. L. Confluence., "Pox wiki.." https://openflow.stanford.edu/display/ONL/POX+Wiki.html, 2015.

[21] Floodlight., "Floodlight sdn openflow controller.." https://github.com/floodlight/floodlight, 2018.

[22] OpenDaylight., "Opendaylight: A linux foundation collaborative project.." https://www.opendaylight.org/, 2021.

[23] RYU, "What's ryu?." https://ryu-sdn.org/, 2022.

[24] D. Borrego Ropero Rafael, Recio Domínguez, "Manual de algorítmica," 05 2017.

[25] K. Appel, W. Haken, and J. Koch, "Every planar map is four colorable. Part II: Reducibility," *Illinois Journal of Mathematics*, vol. 21, no. 3, pp. 491 – 567, 1977.

[26] M. Aslan and N. A. Baykan, "A performance comparison of graph coloring algorithms," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 4, no. Special Issue-1, pp. 1 – 7, 2016.

[27] A. Indian Institute of Information Technology, "Comparison between mamdani and sugeno fuzzy inference system." https://www.geeksforgeeks.org/comparison-between-mamdani-and-sugeno-fuzzy-inference-system/, 2020.

[28] F. Tang, Z. M. Fadlullah, B. Mao, and N. Kato, "An intelligent traffic load prediction-based adaptive channel assignment algorithm in sdn-iot: A deep learning approach," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5141–5154, 2018.

[29] Y. Gupta, A. Saxena, A. Saini, and A. Sharan, "Development of hybrid similarity measure using fuzzy logic for performance improvement of information retrieval system," in *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1–5, 2014.

[30] N. Ul Hasan, W. Ejaz, N. Ejaz, H. S. Kim, A. Anpalagan, and M. Jo, "Network selection and channel allocation for spectrum sharing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 980–992, 2016.

[31] O. Jeunen, P. Bosch, M. V. Herwegen, K. V. Doorselaer, N. Godman, and S. Latré, "A machine learning approach for ieee 802.11 channel allocation," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 28–36, 2018.

[32] H. Tabrizi, G. Farhadi, J. M. Cioffi, and G. Aldabbagh, "Coordinated cognitive tethering in dense wireless areas," *ETRI Journal*, vol. 38, no. 2, pp. 314–325, 2016.

[33] H. M. ., "17 types of similarity and dissimilarity measures used in data science." https://towardsdatascience.com/17-types-of-similarity-and-dissimilarity-measures-used-in-data-science-3eb914d2681, 2021.

[34] Ryu, "Openflow v1.3 messages and structures." https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html, 2020.

[35] M. P. Contributors., "Mininet wifi." https://mininet-wifi.github.io/, 2022.