MITIGACIÓN DE INTERFERENCIAS EN REDES WI-FI DE 2.4 GHz

Realizado por:

Carlos Humberto Romero Diaz.

Nicolas Gallego Quiroga.

Ingeniería de Telecomunicaciones.

Fundación Universitaria Compensar.

Dirigido por:

Fran Edward Perez Ortiz

Sistematización de proyectos de investigación.

Bogotá 2024

Esta obra está bajo una Licencia Creative Commons Atribución-No Comercial-Compartir Igual 4.0 Internacional.

1 Resumen

Las redes Wi-Fi de 2.4 GHz son susceptibles a presentar interferencias, comúnmente evidenciamos esta problemática en entornos domésticos, contando con una alta densidad de dispositivos conectados a las redes. Estas interferencias pueden ser causadas por la superposición de canales, la presencia otros dispositivos utilizando la misma frecuencia, o incluso obstáculos físicos. Afectando de forma directa el rendimiento, provocando velocidades de transmisión de datos más lenta, inestabilidad en la conexión y una experiencia de usuario deficiente. Por lo tanto, este proyecto se centra en la simulación de un entorno virtualizado, técnica de rendimiento en la red Wi-Fi, de acuerdo al uso de canales en la banda 2.4 GHz y, por último, la implementación de un método de inteligencia artificial para elegir el canal con mejor rendimiento, buscando mejorar la calidad y estabilidad de la señal en el entorno doméstico.

Iniciamos con una simulación en NetAnim y ns-3 para poder comprender como funciona una red Wi-Fi y la transmisión de los paquetes entre nodos, lo cual nos proporcionó una base de conocimiento, familiaridad con herramientas de simulación y generación de código para el funcionamiento. A continuación, realizamos un análisis práctico en un entorno doméstico real utilizando herramientas como Nperf, NetSpot e inSSIDer para evaluar el rendimiento de la red, conocer la cobertura y poder determinar el canal óptimo. Identificamos la configuración de la red utilizando la herramienta TP-Link y realizar el cambio manual del canal.

Finalmente, exploramos el uso de algoritmos genéticos, desarrollando un código que analiza múltiples redes y su conexión a los diferentes canales en la banda 2.4 GHz, y después de realizado el análisis informa el canal más óptimo para el equipo.

1.1 Abstract

2.4 GHz Wi-Fi networks are susceptible to interference, a problem commonly seen in home environments with a high density of connected devices. This interference can be caused by overlapping channels, the presence of other devices using the same frequency, or even physical obstacles. This directly affects performance, leading to slower data transmission speeds, connection instability, and a poor user experience. Therefore, this project focuses on simulating a virtualized environment, analyzing Wi-Fi network performance techniques according to the use of channels in the 2.4 GHz band, and finally, implementing an artificial intelligence method to choose the best performing channel, seeking to improve signal quality and stability in the home environment. We started with a simulation in NetAnim and ns-3 to understand how a Wi-Fi network works and how packets are transmitted between nodes. This provided us with a knowledge base, familiarity with simulation tools, and code generation for operation. Next, we performed a practical analysis in a real home environment using tools such as Nperf, NetSpot, and inSSIDer to evaluate network performance, understand coverage, and determine the optimal channel. We identified the network configuration using the TP-Link tool and performed a manual channel change.

Finally, we explored the use of genetic algorithms, developing code that analyzes multiple networks and their connection to different channels in the 2.4 GHz band. After performing the analysis, it reports the most optimal channel for the device.

Tabla de contenido

1 Resumen 2

1.1 Abstract 3

2 Introducción 7

3 Planteamiento del problema y justificación 8

4 Objetivos 9

4.1 Objetivo general 9

4.2 Objetivos específicos 9

5 Alcance 10

6 Limitaciones 10

7 Marco teórico 11

7.1 Redes wifi y su evolución 11

7.2 Desafíos que presenta la red Wi-Fi 2.4 GHz 12

7.3 Algoritmos genéticos 13

7.4 Redes neuronales artificiales 14

8 Método 14

8.1 Entorno virtualizado 14

8.1.1 Herramientas 14

8.1.2 Procedimiento 14

8.2 Análisis entorno real 15

8.2.1 Herramientas 15

8.2.2 Procedimiento 15

8.3 Algoritmo genético 15

8.3.1 Herramientas 15
8.3.2 Procedimiento 16
9 Resultados 16
9.1 Implementación virtualizada 16
9.2 Creación e implementación del código 18
9.3 Implementación entorno real 21
9.4 Implementación del algoritmo genético 26
9.5 Creación e implementación del algoritmo 30
10 Conclusiones 35
11 Bibliografías 36

Índice de tablas

Tabla 2 Evolución de los estándares wifi 12

Índice de figuras

Figura 1 Evolución de los estándares wifi – Creación Propia 11
Figura 2 Topología Base NetAnim - Creación Propia 17
Figura 3 Transmisión de Paquetes - Creación propia 17
Figura 4 Librerías de la configuración inicial - Creación Propia 18
Figura 5 Creación de Nodos y Configuración del Canal – Creación Propia 18
Figura 6 Configuración Wi-Fi y Movilidad – Creación Propia 19
Figura 7 Configuración de Red - Creación Propia 19
Figura 8 Recepción de Paquetes - Creación Propia 20
Figura 9 Envío de paquetes y Simulación - Creación Propia 20
Figura 10 Ejecución de Simulación - Creación Propia 20

Figura 11 Validación del canal en que se encuentra la red 21

Figura 12 Validación configuración del canal - Creación Propia 22

Figura 13 Datos Iniciales Red Wi-Fi 22

Figura 14 Mapa de calor, canal 9 - Creación Propia 23

Figura 15 Cambio de Canal: Automático a Manual - Creación Propia 24

Figura 16 Cambio manual al Canal 11 24

Figura 17 Test después del Cambio en el Canal 25

Figura 18 Mapa de calor, canal 11- Creación Propia 25

Figura 19 Instalación Python 3.13.0 26

Figura 20 Instalación de Librerías Python 3.13.0 27

Figura 21 Creación de Entorno Virtualizado - Creación Propia 27

Figura 22 Documentación en la Carpeta de Simulación - Creación Propia 28

Figura 23 Creación de Carpeta de Simulación - Creación Propia 28

Figura 24 Activación del Entorno - Creación Propia 29

Figura 25 Instalación de Librerías - Creación Propia 29

Figura 26 Configuración de Parámetros - Creación Propia 30

Figura 27 Configuración Inicial del Redes Cercanas - Creación Propia 31

Figura 28 Información Detallada de las Redes Cercanas - Creación Propia 31

Figura 29 Filtro para obtener las Redes en la Banda 2.4 GHz - Creación Propia 32

Figura 30 Comparativa entre los Individuos y los Canales Vecinos - Creación Propia 32

Figura 31 Mutación en los individuos - Creación Propia 33

Figura 32 Generación y Almacenamiento de Script - Creación Propia 34

Figura 33 Funcionamiento del Script - Creación Propia 34

Figura 34 Resultado Final, Canal Idóneo - Creación Propia 34

2 Introducción

En la actualidad la creciente demanda de conectividad inalámbrica ha convertido a las redes Wi-Fi en un servicio esencial en los hogares modernos. Sin embargo, la banda 2.4 GHz, es con diferencia la más utilizada por los equipos de cómputo, teléfonos inteligentes, laptops y electrodomésticos, encontrándose cada vez más congestionada. Esta congestión, provocada por la gran cantidad de redes y la superposición de canales, se refleja en interferencias que no permiten una conexión estable, se reduce la velocidad y genera un servicio que afecta negativamente la experiencia del usuario. Si bien existe métodos para mitigar las interferencias, son muy poco utilizados.

Con este contexto, encaminamos el proyecto en proponer la optimización de las redes Wi-Fi 2.4 GHz en entornos domésticos, mediante un enfoque de entendimiento, desde la implementación en un entorno virtualizado, el análisis experimental y la inteligencia artificial. En un inicio realizamos la construcción de un entorno virtual en NetAnim y ns-3 para simular el comportamiento de una red y evidenciar la comunicación entre nodos, entrega y recepción de paquetes, conociendo el funcionamiento de los programas y familiarizándonos con la generación de pensamiento lógico para la creación de código a partir de las necesidades de la implementación.

Continuamos con un análisis en un entorno real, utilizando herramientas como Nperf, NetSpot e inSSIDer para evidenciar el comportamiento, zonas de mayor congestión y rendimiento de la red, identificando la velocidad en la transmisión de los datos, tanto de bajada como subida, además de ver los canales disponibles y la saturación de dispositivos encontrados. Frente a la información disponible, implementamos el cambio canal de forma manual con la herramienta TP-Link, lo cual nos confirmó una mejora significativa en la velocidad y estabilidad en la red.

Por último, con el objetivo de automatizar el proceso de optimización, exploramos el potencial de los algoritmos genéticos. Desarrollamos un código, que, a través de múltiples análisis de las redes en el entorno donde se encuentra el equipo de cómputo, permite identificar el canal más optimo, mejorando la calidad en la conexión.

3 Planteamiento del problema y justificación

En la actualidad, las redes Wi-Fi constituyen un factor importante en la conectividad inalámbrica en hogares aumento, empresas y espacios públicos. Esto se convierte en una problemática, debido a que el espectro 2.4 GHz, es ampliamente más utilizada, enfrentando un uso excesivo debido a la limitada cantidad de canales y por consecuencia la coexistencia de múltiples dispositivos operando en la misma banda. Este escenario genera interferencias, afectación en el rendimiento, la velocidad y estabilidad de las conexiones [1], representando un desafío técnico y práctico para la experiencia final del usuario.

En nuestro proyecto buscamos abordar esta problemática a partir de una simulación en un entorno virtualizado, análisis experimental y la implementación de un sistema inteligencia basado en algoritmo genético que permita optimizar el rendimiento de la red.

La justificación en nuestro proyecto se encuentra en la necesidad de desarrollar una solución innovadora que permita optimizar el rendimiento de las redes Wi-Fi 2.4 GHz en entornos domésticos, complementando los métodos tradicionales.

4 Objetivos

4.1 Objetivo general

Realizar un entorno virtualizado con un algoritmo genético para mitigar las interferencias en redes Wi-Fi de 2.4 GHz en entornos domésticos, con el fin de mejorar la calidad en la señal y estabilidad en la red.

4.2 Objetivos específicos

I. Implementar un entorno virtualizado para analizar el funcionamiento de una red domestico controlada, compuesto por un nodo principal y tres estaciones con herramientas como NetAnim y ns-3.

II. Analizar el canal en el cual se encuentra la conexión del equipo para evaluar los resultados antes y después de implementar el cambio manual del canal. Utilizando herramientas como NetSpot e inSSIDer, con base en métricas de velocidad de descarga, carga y latencia.

III. Plantear un método de algoritmo genético que sirva como complemento para la técnica de cambio manual en el canal, que permita analizar un punto de vista basado en la repetición, para una conexión estable y sin interferencias en la red 2.4 GHz de forma automática.

5 Alcance

En nuestro proyecto nos enfocamos en optimización del rendimiento de redes Wi-Fi de 2.4 GHz en entorno doméstico, buscando mejorar la calidad y estabilidad de la conexión mediante diferentes estrategias.

I. Simulación virtualizada: Utilizamos herramientas NetAnim y ns-3, que nos permitieron visualizar el comportamiento de una red Wi-Fi en un entorno controlado, evidenciando el envío y recepción de paquetes. Además de conocer el funcionamiento de las herramientas utilizadas.

II. Entorno domestico real: Realizamos un análisis de una red Wi-Fi doméstica, con herramientas como NetSpot, inSSIDer y Nperf para medir la calidad en la conexión y visualizar zonas con mala cobertura. Realizamos un cambio manual en el canal, evidenciando mejora frente al antes y después en el cambio de frente a un equipo de cómputo específico.

III. Algoritmo genético: Implementamos un algoritmo genético que nos permite complementar métodos tradicionales, analizando el espectro de frecuencias e identificando el canal con mejor cobertura y menor solapamiento para un mejor funcionamiento del equipo en el que se ejecuta la implementación.

6 Limitaciones

I. La simulación realizada en un entorno virtualizado, nos permite una visión y el conocimiento técnico en el funcionamiento y aplicación de los diferentes programas, además de acercarnos al pensamiento lógico con la generación de código, pero no replica con exactitud todas las variables que se presentan en un entorno doméstico real, como la intensidad de la señal, la presencia de obstáculos físicos y la interferencia de otros dispositivos.

II. En cuanto a la implementación en el entorno real, nos permite evidenciar la mejoría en el cambio del canal, sin embargo, debido al tiempo, no pudimos explorar otras técnicas lo cual nos genera un sesgo en el funcionamiento.

III. La implementación del algoritmo genético no está diseñada para detectar o mitigar interferencias en la red Wi-Fi, ni detectar elementos físicos que impidan una mejoría en la red.

7 Marco teórico

7.1 Redes wifi y su evolución

El estándar IEEE 802.11 creado en el año 1997 fue la base para la conectividad inalámbrica con una velocidad inicial de 2 Mbps, desde ese año las redes wifi evolucionaron mediante varias versiones para mejorar la velocidad, el alcance y la capacidad de la red, el estándar IEEE .11n fue clave para el wifi ya que logro una velocidad de 600 Mbps todo esto operando en la banda 2.4 GHz[2], el cual permitió múltiples antenas para transmitir múltiples flujos de datos simultáneamente, con esta evolución se remplazó en gran mayoría las redes cableadas. En el año 2013 surgió el estándar IEEE 802.11ac donde fue una gran evolución ofreciendo velocidades hasta 3.5 Gbps operando en la banda 5 GHz la cual contiene más canales y mayor ancho de banda.[3]

A continuación, se muestra una tabla donde se detalla a fondo la evolución de los estándares de la red wifi.

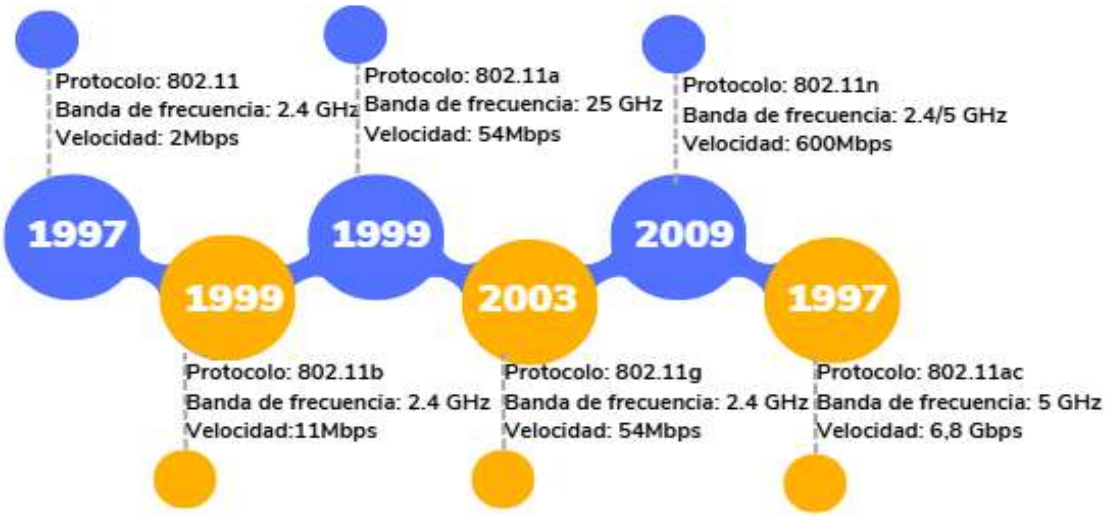


Figura 1 Evolución de los estándares wifi – Creación Propia

7.2 Desafíos que presenta la red Wi-Fi 2.4 GHz

La banda 2.4 GHz sigue siendo utilizada en entornos domésticos ya que es una de las bandas que tiene mayor alcance comparada con la banda 5 GHz, pero el inconveniente de esta banda es que es más propensa a solapamientos de canales, esto conlleva a interferencias y lentitud en la red en áreas con alto número de dispositivos conectados[4].

El solapamiento de canales es un tipo de interferencia particularmente común y perjudicial en las redes Wi-Fi. Este fenómeno ocurre cuando dos o más redes Wi-Fi dentro del mismo rango de alcance transmiten datos utilizando el mismo canal o canales cercanos dentro de la misma banda de frecuencia.

Las interferencias se producen cuando dos o más redes Wi-Fi en proximidad transmiten datos en el mismo canal. Este tipo de interferencia es particularmente común en la banda de 2.4 GHz, donde la disponibilidad limitada de canales no superpuestos (solo los canales 1, 6 y 11) aumenta la probabilidad de que múltiples redes vecinas operen en la misma frecuencia[5].

En la siguiente tabla se muestra el ancho de banda de cada canal, donde se evidencia que en cada canal tiene solapamiento de los canales vecinos por su ancho de banda, cada canal ocupa un ancho de banda de 22 MHz.

Canal	Frecuencia baja	Frecuencia alta
1	2.412 GHz	2.434 GHz
2	2.417 GHz	2.439 GHz
3	2.422 GHz	2.444 GHz
4	2.427 GHz	2.449 GHz
5	2.432 GHz	2.454 GHz
6	2.437 GHz	2.459 GHz
7	2.442 GHz	2.464 GHz
8	2.447 GHz	2.469 GHz
9	2.452 GHz	2.474 GHz
10	2.457 GHz	2.479 GHz
11	2.462 GHz	2.484 GHz

Tabla 1 Evolución de los estándares wifi

Por ejemplo, se tiene el canal 1 que tiene el rango de frecuencias entre 2.412 GHz a 2434 GHz, el canal 2 comienza en la frecuencia 2.417 GHz y termina en la frecuencia 2.439 GHz lo que significa que el canal 2

se superpone con el canal 1 en las frecuencias de 2.417 GHz a 2434 GHz, este caso se repite en la mayoría de los canales vecinos, los canales que no se superponen son los canales 1, 6 y 11 como se muestra a en la **Figura 1**.

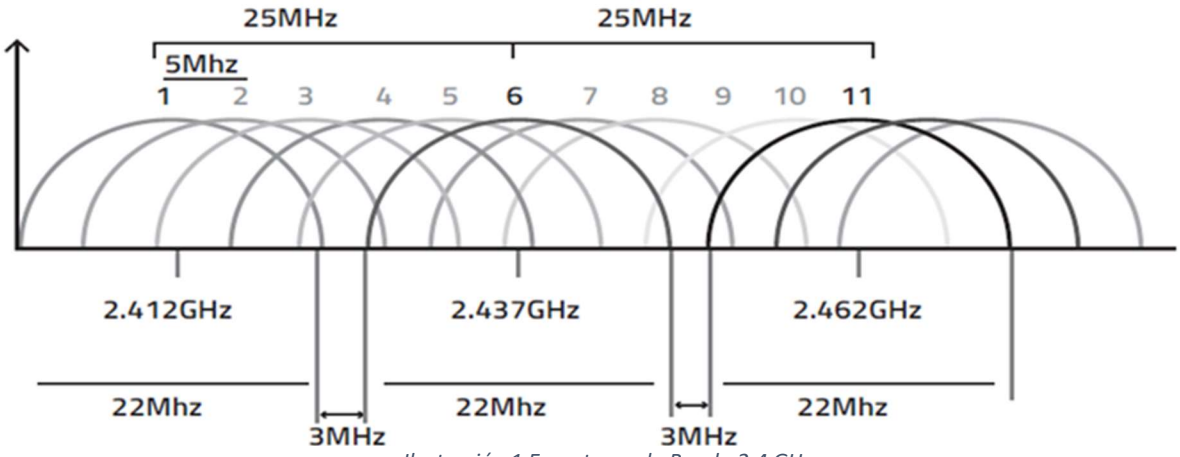


Ilustración 1 Espectro en la Banda 2.4 GHz

7.3 Algoritmos genéticos

Los algoritmos genéticos es un método de búsqueda basado en la evolución biológica en el cual genera varias soluciones[6], después escoge las mejores para reproducirlas creando una nueva población para repetir el proceso hasta llegar a la mejor solución, los algoritmos genéticos pueden aplicarse en las siguientes áreas.[7]

- ✓ Diseño de ingeniería
- ✓ Aprendizaje automático
- ✓ Economía
- ✓ Asignación de procesos en redes en redes.[8]

I. Ventajas de los algoritmos genéticos

- ✓ Permiten explorar varias soluciones a la vez
- ✓ Son aplicables para la gran mayoría de problemas
- ✓ No requiere información específica del problema a resolver.

II. Desventajas

- ✓ Pueden llegar a ser costoso en la implementación para problemas grandes
- ✓ No está garantizada la convergencia.

7.4 Redes neuronales artificiales

Las redes neuronales artificiales se utilizan para resolver problemas complejos, las cuales son inspiradas en la estructura del cerebro humano[9]. Estas redes son muy útiles para gestionar tareas de clasificación predicción y optimización. Está compuesta por neuronas artificiales que se organizan por capas conectándose entre sí, donde las conexiones tienen pesos que se ajustan mediante el proceso de aprendizaje para minimizar errores y mejorar la precisión de la red[6].

Las redes neuronales artificiales tienen dos tipos de aprendizaje[10], supervisado y no supervisado. En el aprendizaje supervisado se usa un conjunto de datos con entradas y salidas conocidas, mientras el aprendizaje no supervisado la misma red debe encontrar patrones para su aprendizaje, es decir, es un aprendizaje autónomo[9], [11].

8 Método

En la implementación del proyecto en la optimización de las redes Wi-Fi de 2.4 GHz en entorno doméstico, tomamos tres enfoques complementarios: simulación en entorno virtual, análisis en un entorno real y algoritmo genético que complemento de los métodos tradicionales.

8.1 Entorno virtualizado

8.1.1 Herramientas

I. ns-3: Es un simulador de redes que nos permite modelar diferentes tipos de redes Wi-Fi.

II. NetAnim: nos permite visualizar las implementaciones realizadas con ns-3.

8.1.2 Procedimiento

Creamos un entorno virtual que representa una red doméstica, incluimos un nodo y tres estaciones (un portátil y dos celulares), definimos los parámetros de la simulación en un código base generado para las necesidades de la implementación, como el número de nodos, la distancia entre ellos, posicionamiento en NetAnim. Finalmente, utilizamos ns-3 compilando nuestro código en formato (.xml), para reconocimiento

y simulación en NetAnim, pudiendo visualizar la comunicación entre nodos y el envío y recepción de paquetes.

8.2 Análisis entorno real

8.2.1 Herramientas

I. **NetSpot:** Software de análisis, nos permite visualizar la cobertura de la red en el entorno doméstico, realizar mapas de calor e identificar zonas con problemas de señal.

II. **inSSIDer:** Herramienta que nos permite visualizar las redes Wi-Fi que se encuentran en el entorno, y poder identificar superposición en los canales.

III. **Nperf:** Nos permite identificar aspectos clave frente a la velocidad del internet, descarga, carga y latencia.

IV. **TP-Link:** Herramienta que nos permite evidenciar parámetros pre establecidos en el canal y modificar a conveniencia.

8.2.2 Procedimiento

Hicimos un análisis en un entorno real. Utilizando las herramientas NetSpot, inSSIDer y Nperf para medir el alcance de la red en diferentes puntos del hogar, analizar la calidad de la señal, a la velocidad en descarga, carga y latencia. Por último, realizamos cambio del canal y evaluamos antes y después frente al comportamiento de la red.

8.3 Algoritmo genético

8.3.1 Herramientas

I. **Python:** Lenguaje de programación que, al instalar las librerías correspondientes, nos permitió generar un script para procesar la información del entorno en la red Wi-Fi y a su vez evaluar el rendimiento.

II. **CMD:** Esta herramienta que viene implementada en el sistema operativo Windows, nos permite ejecutar, y con los comandos interactuar para poder ejecutar el algoritmo genético.

8.3.2 Procedimiento

Definimos la estructura del algoritmo genético, incluyendo los canales disponibles, generaciones de las redes vecinas y la generación de la simulación[9]. Obteniendo como resultado el canal más eficiente de acuerdo al análisis de conexión en los canales con las redes vecinas.

9 Resultados

9.1 Implementación virtualizada

Realizamos la implementación de una topología representativa de un entorno doméstico, compuesta por un nodo central (router) y tres estaciones (computador personal, dos celulares), representando los dispositivos finales conectados a la red. Este diseño fue construido en un entorno virtualizado utilizando VirtualBox, una herramienta que nos permite aprovechar los recursos del equipo de cómputo personal para la simulación. Elegimos Ubuntu 20.04 [12] como sistema operativo debido a su estabilidad comprobada con las herramientas utilizadas.

El simulador ns-3 en su versión 3.34, se instaló y configuró como el núcleo de la simulación. Esta herramienta nos permitió modelar la red, los tiempos de transmisión y los tiempos correspondientes al envío y recepción de paquetes[2]. Posteriormente, generamos un código acorde a las necesidades del proyecto, definimos la cantidad de paquetes enviados (50 paquetes configurados), los parámetros de cada enlace **Figura 5**, y la ubicación virtual de cada dispositivo dentro de la red **Figura 6.** El tiempo de recepción de los paquetes se estableció en milisegundos para reflejar la latencia esperada en una red doméstica estándar **Figura 9,** en la que los tiempos de respuesta suelen estar en este rango **Figura 3.**

Para facilitar la interpretación de la lectura en los datos simulados, integramos NetAnim en su versión 3.108[13], como herramienta de visualización. Esta se conecta con ns-3 mediante un archivo de animación (.anim) generado en el simulador, conteniendo la información sobre la posición de los nodos, el tráfico de paquetes y los tiempos asociados, permitiendo graficar y analizar visualmente el comportamiento de la red implementada en tiempo real.

Finalmente logramos implementar con éxito la topología planteada y obtener los resultados esperados. En la simulación podemos ver la transmisión efectiva de los paquetes establecidos entre los nodos, con un tiempo promedio de 1.2 ms por paquete **Figura 3,** datos consistentes con las redes domésticas.

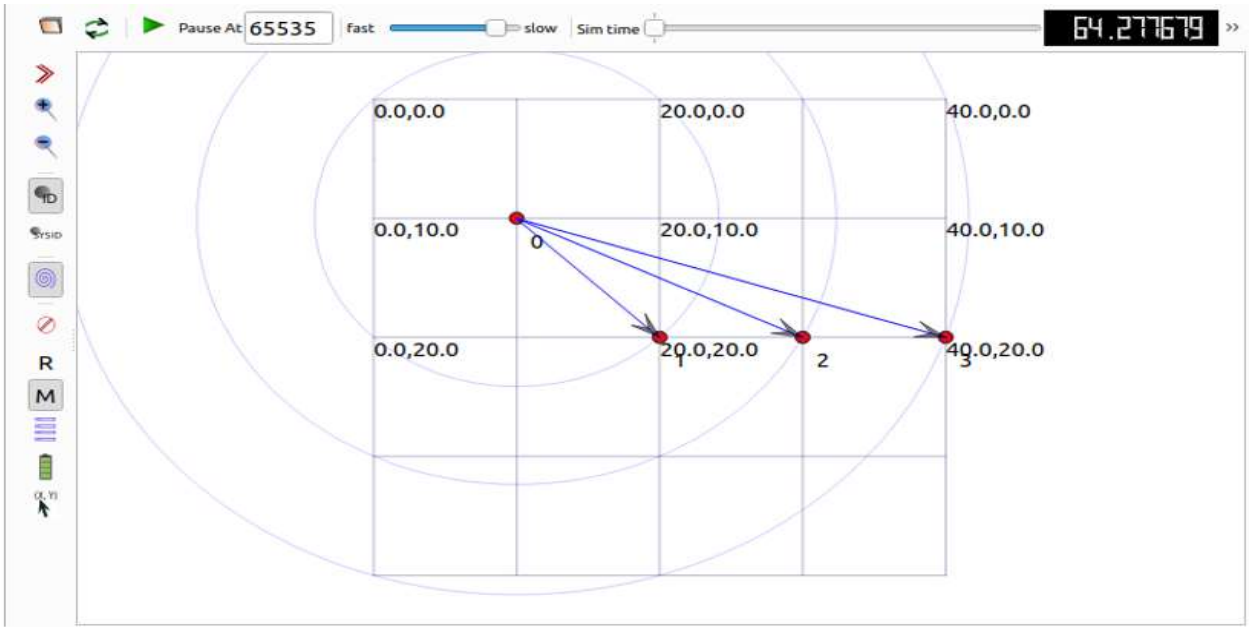


Figura 2 Topología Base NetAnim - Creación Propia

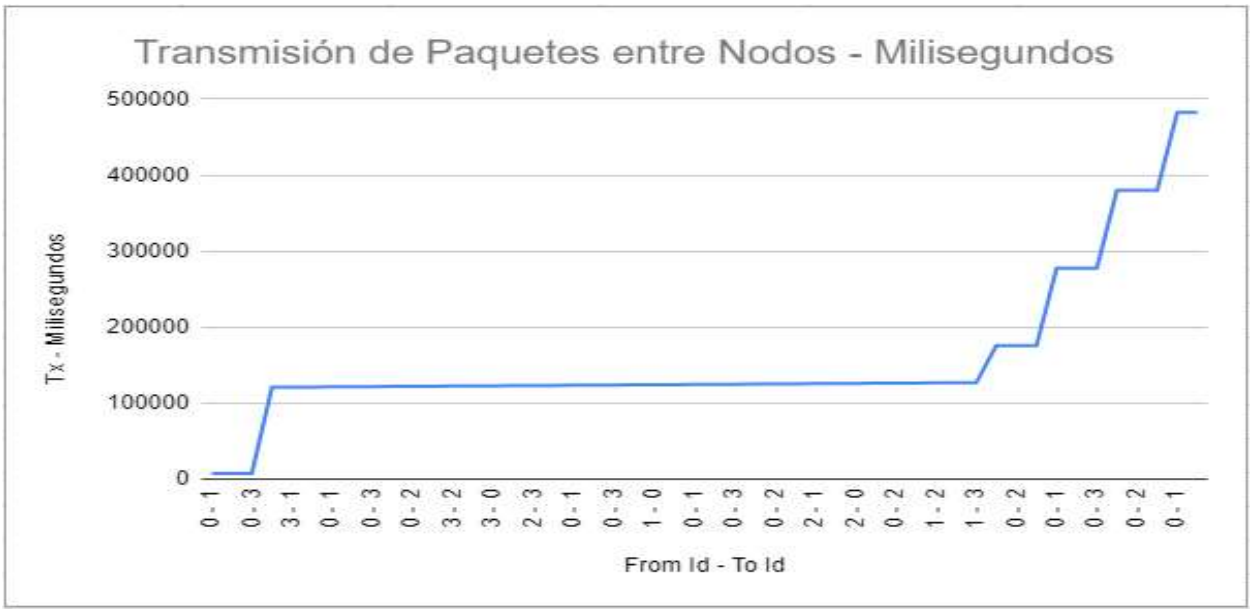


Figura 3 Transmisión de Paquetes - Creación propia

9.2 Creación e implementación del código

En esta primera configuración del código, incluimos las librerías que necesitamos en ns-3.34 para realizar la simulación de la red Wi-Fi, poder gestionar los canales y configurar la movilidad de los nodos designados. También incluimos el módulo NetAnim para poder generar el archivo (.xml) para la correcta simulación.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/wifi-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/packet-sink.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;
```

Figura 4 Librerías de la configuración inicial - Creación Propia

Continuamos con la configuración del punto de acceso y tres estaciones, representando cualquier dispositivo que se pueda conectar a la red. Procedemos a configurar para que la señal Wi-Fi se propague.

```
NodeContainer wifiApNode;
wifiApNode.Create(1);

NodeContainer wifiStaNodes;
wifiStaNodes.Create(3);  // Tres estaciones
}

YansWifiChannelHelper channel;
channel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
channel.AddPropagationLoss("ns3::FriisPropagationLossModel");

YansWifiPhyHelper phy;
phy.SetChannel(channel.Create());
```

Figura 5 Creación de Nodos y Configuración del Canal – Creación Propia

Generamos la configuración en base a estándares de Telecomunicaciones: 802.11g. Definimos el tipo de MAC para el punto de acceso y las estaciones. Definimos la posición y movilidad de los nodos en la simulación.

```
WifiHelper wifi;
wifi.SetStandard(WIFI_STANDARD_80211g);

WifiMacHelper mac;
Ssid ssid = Ssid("ns-3-wifi");
mac.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
NetDeviceContainer apDevice = wifi.Install(phy, mac, wifiApNode);

mac.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid));
NetDeviceContainer staDevices = wifi.Install(phy, mac, wifiStaNodes);

MobilityHelper mobility;
mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                              "MinX", StringValue("0.0"),
                              "MinY", StringValue("0.0"),
                              "DeltaX", StringValue("5.0"),
                              "DeltaY", StringValue("5.0"),
                              "GridWidth", StringValue("3"),
                              "LayoutType", StringValue("RowFirst"));

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(wifiApNode);
mobility.Install(wifiStaNodes);
```

Figura 6 Configuración Wi-Fi y Movilidad – Creación Propia

Definimos la instalación de los protocolos de internet y asignamos dirección IP.

```
InternetStackHelper stack;
stack.Install(wifiApNode);
stack.Install(wifiStaNodes);

Ipv4AddressHelper address;
address.SetBase("192.168.1.0", "255.255.255.0");
Ipv4InterfaceContainer staInterfaces = address.Assign(staDevices);
Ipv4InterfaceContainer apInterfaces = address.Assign(apDevice);
```

Figura 7 Configuración de Red - Creación Propia

Configuramos el punto de acceso para que pueda actuar como receptor de paquetes.

```
uint16_t port = 9; // puerto de escucha
PacketSinkHelper sink("ns3::UdpSocketFactory", InetSocketAddress(Ipv4Address::GetAny(), port));
ApplicationContainer sinkApps = sink.Install(wifiApNode);
sinkApps.Start(Seconds(0.0));
sinkApps.Stop(Seconds(10.0));
```

Figura 8 Recepción de Paquetes - Creación Propia

Aquí creamos una aplicación que envié paquetes de forma constante desde las estaciones al punto de acceso. Configuramos la visualización de la simulación en NetAnim 3.34, definiendo en posiciones fijas el punto de acceso y estaciones.

```
OnOffHelper onoff("ns3::UdpSocketFactory", InetSocketAddress(apInterfaces.GetAddress(0), port));
onoff.SetConstantRate(DataRate("50kb/s")); // Tasa de transmisión
ApplicationContainer apps = onoff.Install(wifiStaNodes);
apps.Start(Seconds(1.0));
apps.Stop(Seconds(10.0));

AnimationInterface anim("wifi-simulation.xml");
anim.SetConstantPosition(wifiApNode.Get(0), 10.0, 10.0); // AP en (10,10)
anim.SetConstantPosition(wifiStaNodes.Get(0), 20.0, 20.0); // Estación 1 en (20,20)
anim.SetConstantPosition(wifiStaNodes.Get(1), 30.0, 20.0); // Estación 2 en (30,20)
anim.SetConstantPosition(wifiStaNodes.Get(2), 40.0, 20.0); // Estación 3 en (40,20)
```

Figura 9 Envío de paquetes y Simulación - Creación Propia

Por último, tenemos la ejecución y finalización.

```
  // Ejecutar la simulación
  Simulator::Run();
  Simulator::Destroy();
  return 0;
}
```

Figura 10 Ejecución de Simulación - Creación Propia

9.3 Implementación entorno real

Con el fin de completar la simulación realizada en el entorno virtualizado, llevamos a cabo un análisis práctico en un entorno domestico real, utilizando herramientas accesibles para los usuarios. Nperf nos permite monitorear, mejorar y conocer la velocidad de descarga, subida y latencia, ayudándonos a identificar y optimizar la red. NetSpot, nos permite realizar un mapeo generalizado de la red en tiempo real, para identificar las áreas con mala cobertura. inSSIDer nos permitió analizar el espectro Wi-Fi, detectar interferencias y evidenciar en qué canal se encuentra la red seleccionada. Por último, utilizamos TP-Link para acceder a la configuración del canal y realizar el ajuste manual para optimizar el rendimiento de la red. Estas herramientas en nuestro proyecto nos permiten conocer el antes y después de los cambios.

Nos centramos en la red WIFI de 2.4 GHz nombrada *OFFLINE*. Primero accedimos con inSSIDer identificando que el canal estaba configurado en el canal 9. Esta validación nos permitió verificar interferencias con otras redes cercanas en el mismo espectro, contribuyendo a un rendimiento inadecuado **Figura 11.**

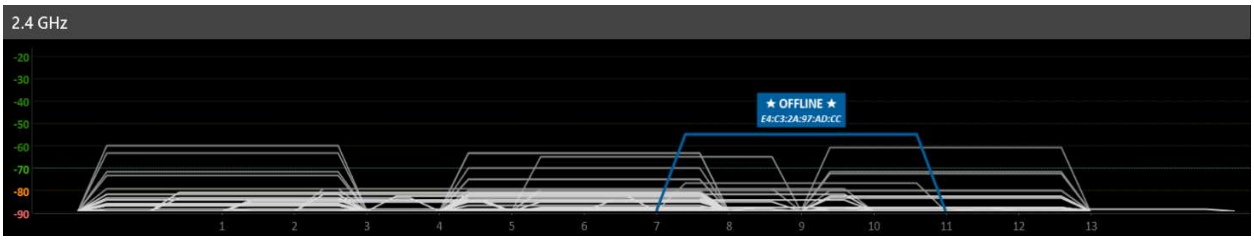


Figura 11 Validación del canal en que se encuentra la red

Hacemos uso de la herramienta TP-Link, identificando propiedades básicas de la conexión. Como evidenciamos en la imagen **Figura 12,** la red opera en modo estándar con seguridad WPA-WPA2, empleando cifrado robusto para la protección de los datos transmitidos. El canal y ancho de banda se encuentran en modo automático. Esta configuración, aunque es adecuada, puede generar interferencias debido a la selección de canales, lo cual no es lo mejor para una red que se encuentra en un entorno con múltiples redes cercanas.



Figura 12 Validación configuración del canal - Creación Propia

Antes de realizar algún cambio, utilizamos Nperf para evaluar el rendimiento actual de la red Wi-Fi *OFFLINE.* La herramienta nos permitió medir los valores de descarga, carga y latencia, proporcionados en base al desempeño de la red en su configuración estándar. Datos fundamentales para poder hacer una comparación y analizar las mejoras.



Figura 13 Datos Iniciales Red Wi-Fi

Realizamos un mapa de calor del entorno domestico real de la red Wi-Fi *OFFLINE*. Con los datos obtenidos con la herramienta NetSpot, podemos identificar las áreas con mejor señal: habitación principal y cocina **Figura 14,** lo que se relacionando directamente con un mejor rendimiento en cuanto a descarga, carga y latencia **Figura 13.**

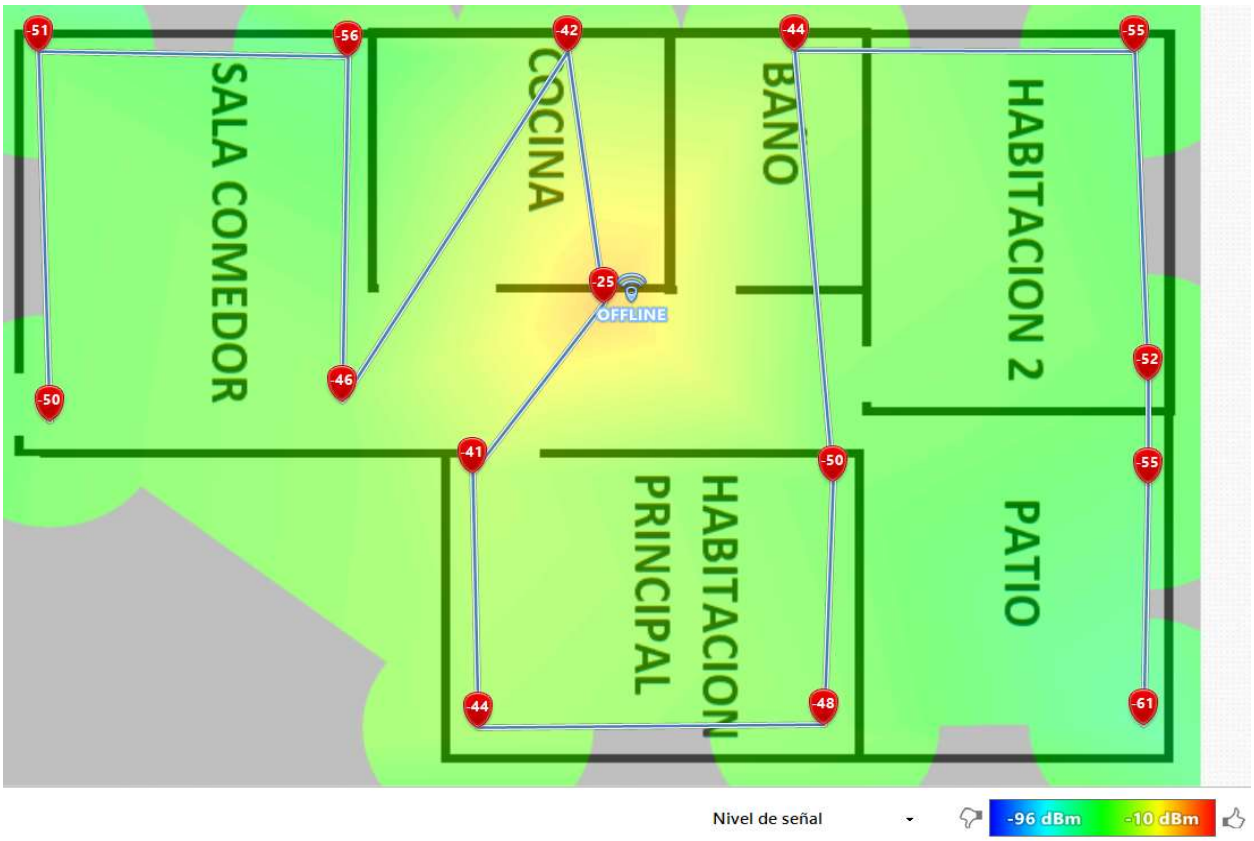


Figura 14 Mapa de calor, canal 9 - Creación Propia

Al comprobar que la red en el canal 9 presenta interferencias debido a redes cercanas operando en el mismo espectro **Figura 11.** Realizamos el cambio manual al canal 11 **Figura 15,** ya que es uno de los pocos canales que no presenta solapamiento en la banda 2.4 GHz, lo que reduciría las interferencias y mejora la estabilidad y conexión de la red.

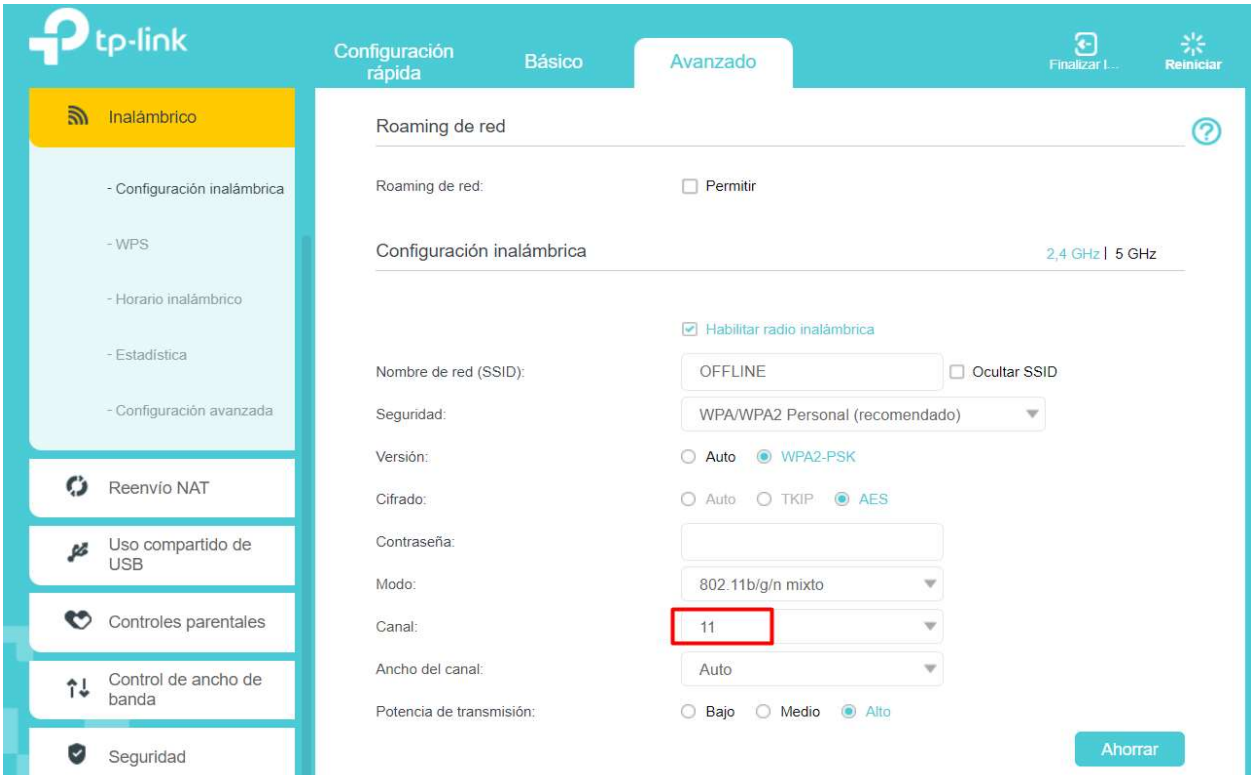


Figura 15 Cambio de Canal: Automático a Manual - Creación Propia

Verificamos el cambio realizado en la red Wi-Fi 2.4 GHz llamada *OFFLINE*, evidenciando el cambio al canal 11. Este cambio nos permite evitar el solapamiento que habíamos identificado en el canal 9 **Figura11,** reduciendo las interferencias y optimizando la estabilidad en la red.

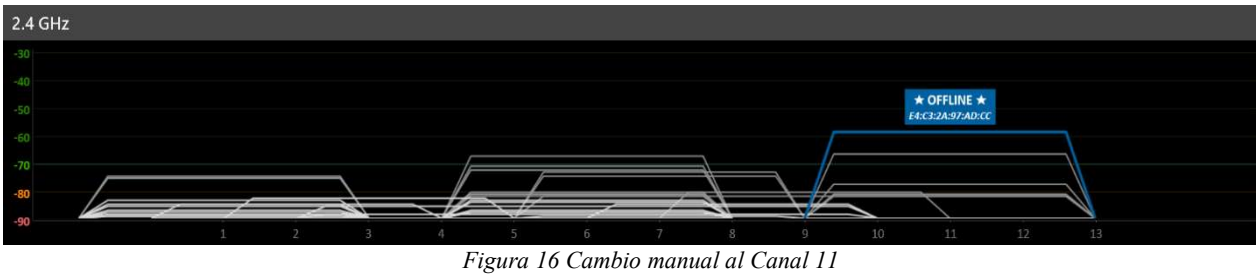


Figura 16 Cambio manual al Canal 11

Después de realiza el cambio de canal, verificamos el rendimiento de la red utilizando Nperf. Evidenciando una mejoría en la velocidad de descarga 26.60 Mb/s y de carga 27.11 Mb/s. A pesar de un leve aumento en latencia 49.20 ms, el cambio ha optimizado de manera significativa el rendimiento general en la red.



Figura 17 Test después del Cambio en el Canal

Después de realizar el cambio del canal manualmente identificamos una mejoría generalizada en las áreas comunes del entorno doméstico, en especial cocina, habitación principal y habitación 2.

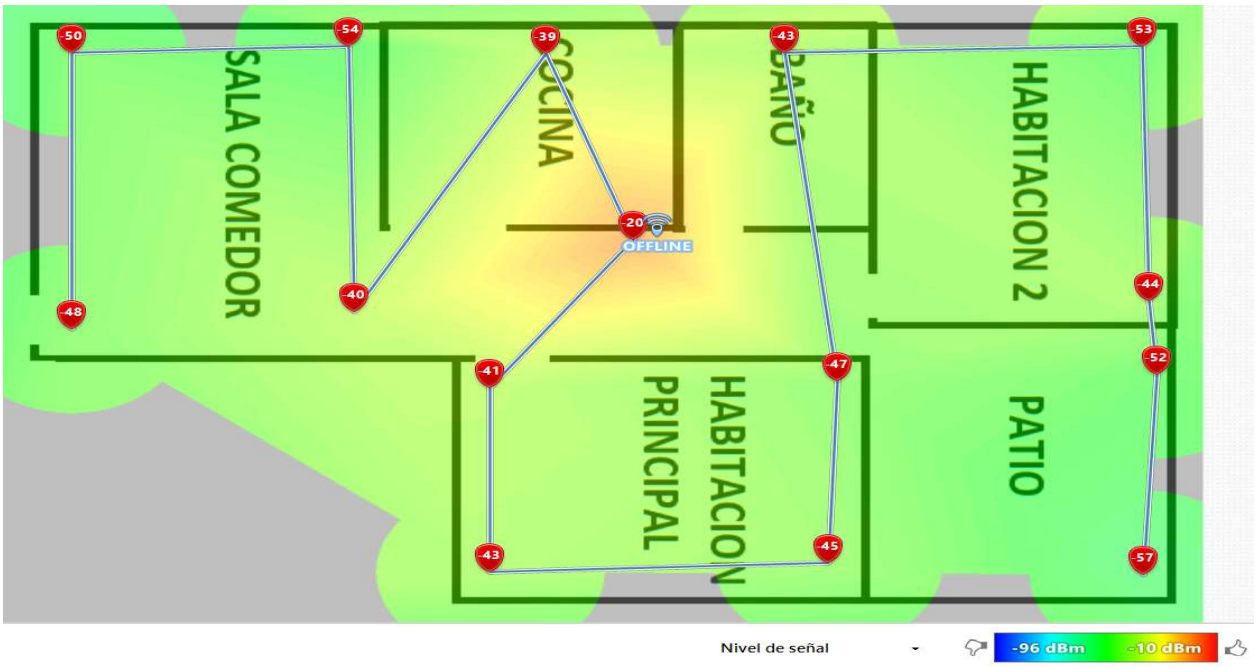


Figura 18 Mapa de calor, canal 11- Creación Propia

9.4 Implementación del algoritmo genético

Para este planteamiento se eligió el algoritmo genético ya que es la mejor opción porque es adecuado para problemas de optimización combinatoria ya que pueden explorar varias soluciones y escoger la más optima.

También porque los algoritmos genéticos solo necesitan evaluar la aptitud de cada individuo, en este caso la configuración de los canales.

Son muy robustos ya que pueden manejar ruido y errores en los datos, esto es muy importante para este caso ya que la información de las redes vecinas puede no ser completa.

Su implementación es más flexible a comparación de otras inteligencias artificiales.

Para la implementación de este algoritmo, se utilizó Python con su más reciente versión 3.13.0 en Windows 11.



Figura 19 Instalación Python 3.13.0

Se marca la opción Add Python to PATH para que Python se pueda ejecutar desde cualquier ubicación en la línea de comandos.

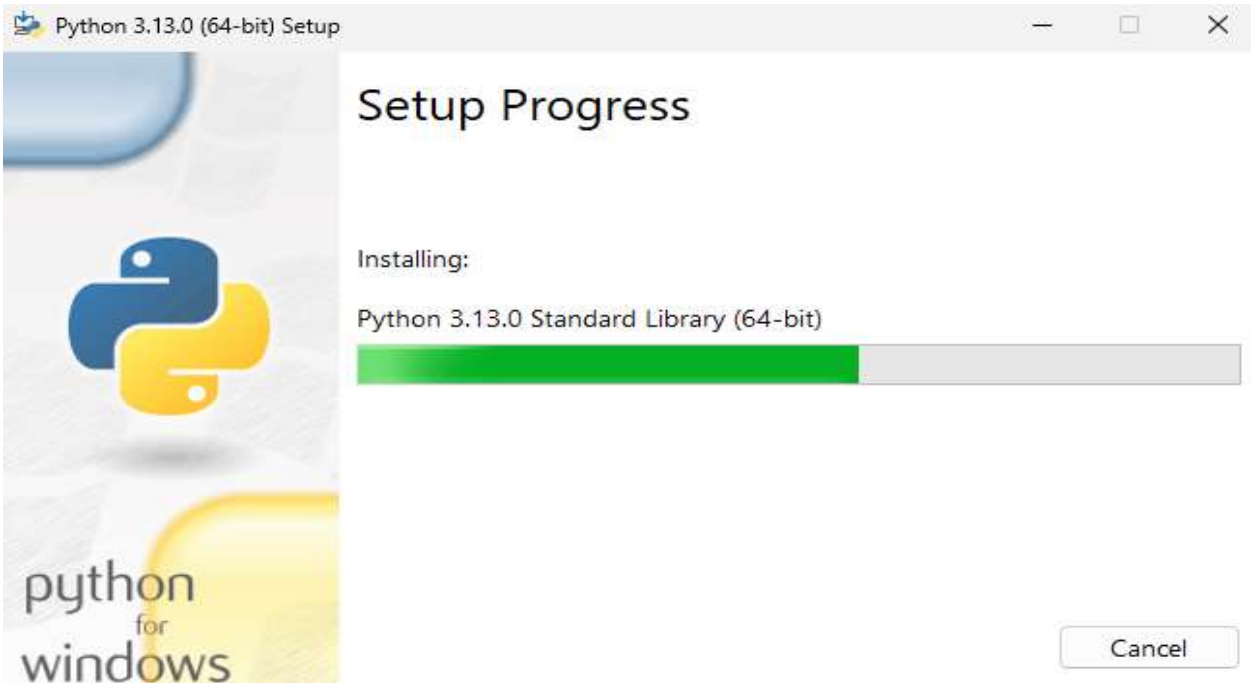


Figura 20 Instalación de Librerías Python 3.13.0

Se debe crear un entorno virtual para evitar problemas de compatibilidad y tener una mejor organización, este entorno lo podemos crear en cualquier carpeta de nuestro sistema, en este caso se creó una carpeta en documentos llamada PROYECTO y dentro de esa carpeta se creó el entorno virtual.

En CMD ingresamos hasta la carpeta PROYECTO y creamos el entorno virtual llamado simulación con el siguiente comando. **Figura 21.**

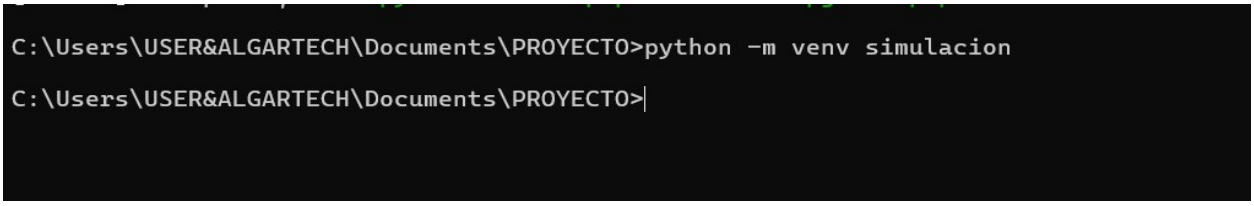


Figura 21 Creación de Entorno Virtualizado - Creación Propia

Cuando creamos el entorno virtual se crea una carpeta llamada simulación como se evidencia en la figura **Figura 23**.

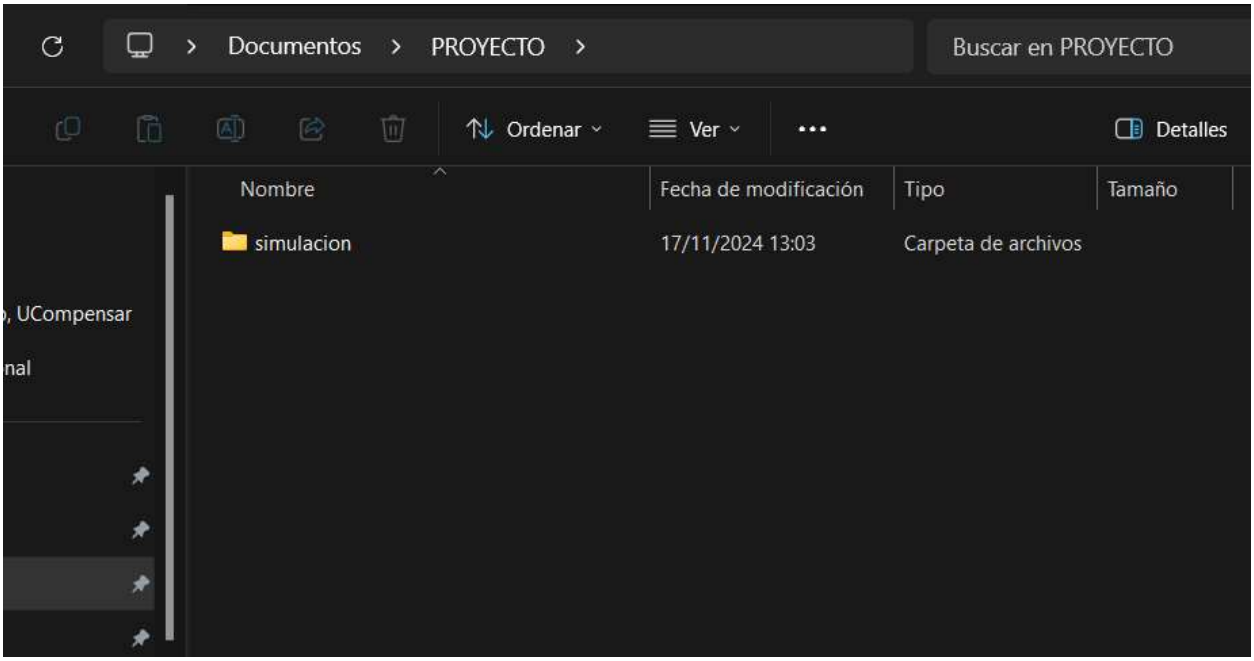


Figura 23 Creación de Carpeta de Simulación - Creación Propia

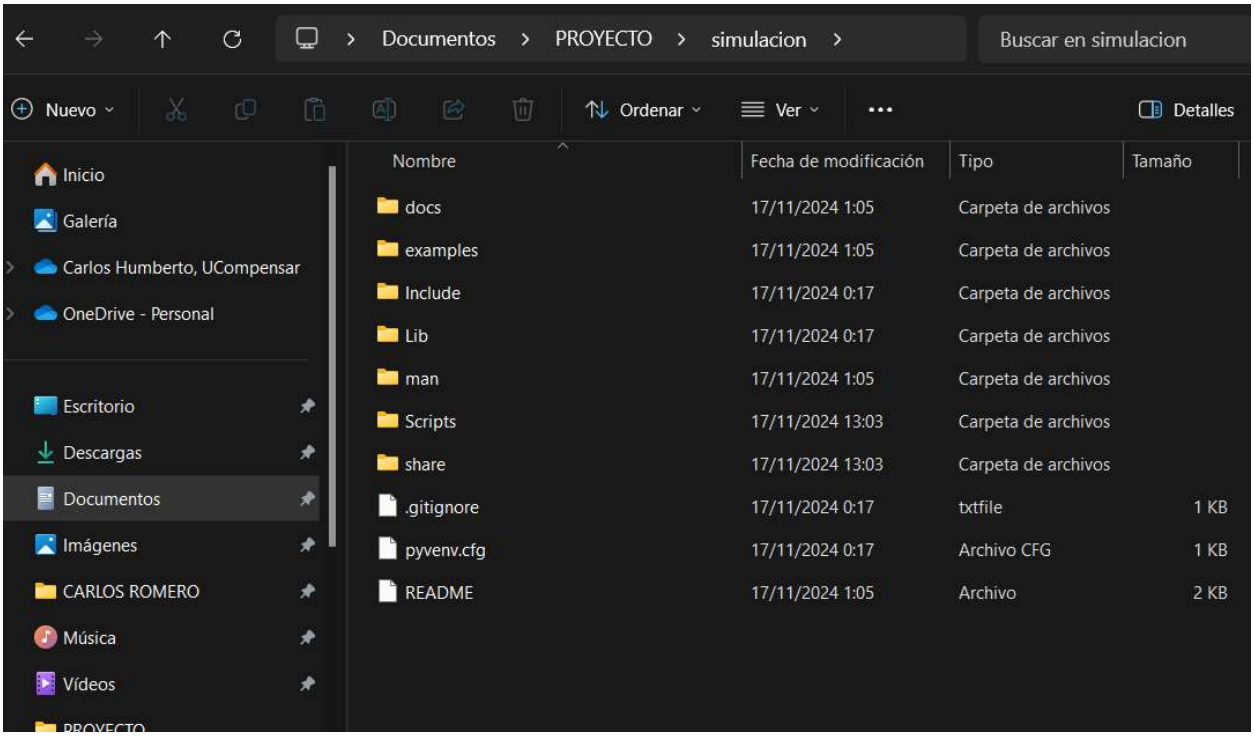


Figura 22 Documentación en la Carpeta de Simulación - Creación Propia

Una vez que creamos el entorno virtual ingresamos a la carpeta creada llamada simulación mediante cmd, también ingresamos a la carpeta Scripts y activamos el entorno.

```
C:\Users\USER&ALGARTECH\Documents\PROYECTO>simulacion\Scripts\activate
El sistema no puede encontrar la ruta especificada.
"ALGARTECH\AppData\Local\Programs\Python\Python313\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Python313\Scripts\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Python313\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Launcher\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
El sistema no puede encontrar la ruta especificada.
"ALGARTECH\AppData\Local\Programs\Python\Python313\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Python313\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Python313\Scripts\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Python313\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
"ALGARTECH\AppData\Local\Programs\Python\Launcher\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
El sistema no puede encontrar la ruta especificada.
"ALGARTECH\AppData\Local\Programs\Python\Python313\" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

(simulacion) C:\Users\USER&ALGARTECH\Documents\PROYECTO>
```

Figura 24 Activación del Entorno - Creación Propia

Una vez activado el entorno se debe instalar las librerías DEAP para algoritmos genéticos, NumPy para cálculos numéricos y matplotlib para la visualización de datos.

```
(simulacion) C:\Users\USER&ALGARTECH\Documents\PROYECTO\simulacion\Scripts>pip install deap numpy matplotlib
Collecting deap
  Downloading deap-1.4.1.tar.gz (1.1 MB)
     ---------------------------------------- 1.1/1.1 MB 6.1 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting numpy
  Downloading numpy-2.1.3-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting matplotlib
  Using cached matplotlib-3.9.2-cp313-cp313-win_amd64.whl.metadata (11 kB)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (4.55.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from matplotlib) (2.9.0.post
0)
Requirement already satisfied: six>=1.5 in c:\users\user&algartech\documents\proyecto\simulacion\lib\site-packages (from python-dateutil>=2.7->matplotlib) (
1.16.0)
Downloading numpy-2.1.3-cp313-cp313-win_amd64.whl (12.6 MB)
     ---------------------------------------- 12.6/12.6 MB 5.7 MB/s eta 0:00:00
Using cached matplotlib-3.9.2-cp313-cp313-win_amd64.whl (7.8 MB)
Building wheels for collected packages: deap
  Building wheel for deap (pyproject.toml) ... done
  Created wheel for deap: filename=deap-1.4.1-py3-none-any.whl size=97350 sha256=424e96a5f95e2d2809d900f8f347b1a0185fe6c38c5387b05fe267bdd73efea0
  Stored in directory: c:\users\user&algartech\appdata\local\pip\cache\wheels\98\de\a4\b2c8e0678e31443ca37d09c3d9d7578c1a119c5fad9fb56841
Successfully built deap
Installing collected packages: numpy, deap, matplotlib
Successfully installed deap-1.4.1 matplotlib-3.9.2 numpy-2.1.3

(simulacion) C:\Users\USER&ALGARTECH\Documents\PROYECTO\simulacion\Scripts>
```

Figura 25 Instalación de Librerías - Creación Propia

9.5 Creación e implementación del algoritmo

Primero establecemos los tipos de datos que usaremos. Configuramos 1 solo punto de acceso. Los canales disponibles (1, 6, 11) para nuestro punto de acceso (AP), seleccionados en específico debido a que no generan solapamiento en la banda 2.4 GHz. También definimos la población a evaluar, inicialmente 20 configuraciones que son generaciones aleatorias para los canales disponibles del AP. 50 ciclos corresponden al número de veces que el algoritmo intentará probar y mejorar las soluciones iniciales de las 20 configuraciones, para escoger el mejor canal. El algoritmo evalúa a cada individuo para medir su configuración (fitness), indicando cual canal minimiza las interferencias con las redes vecinas. Máxima (ambas en el canal 4), media (6 y 7) o mínima (si están separados por más de 2 canales) y escoge el canal con menor interferencias en comparación con las redes vecinas, contando con mayor aptitud.

```
import random
import subprocess
import re
from deap import base, creator, tools, algorithms


num_aps = 1
canales = [1, 6, 11]
tamaño_configuraciones = 20
ciclos = 50
probabilidad_cambio_canal = 0.1


creator.create("InterferenciaMin", base.Fitness, weights=(-1.0,))
creator.create("ConfiguracionAP", list, fitness=creator.InterferenciaMin)

toolbox = base.Toolbox()
toolbox.register("canal_asignado", random.choice, canales)
toolbox.register("configuracion_ap", tools.initRepeat, creator.ConfiguracionAP,
toolbox.canal_asignado, n=num_aps)
toolbox.register("configuraciones_red", tools.initRepeat, list, toolbox.configuracion_ap)
```

Figura 26 Configuración de Parámetros[14]

Continuamos con el comando netsh propio del uso en Windows, con el podemos obtener información de las redes cercanas, detalles de la configuración en la red y los nombres, convirtiendo esta información en datos legibles. Si ocurre algún error, nos devuelve un mensaje (None) indicando que no se obtuvo resultados.

```
def ejecutar_comando_netsh():
    comando = "netsh wlan show networks mode=bssid"
    try:
        resultado = subprocess.check_output(comando, shell=True,
stderr=subprocess.STDOUT).decode("cp850")
    except subprocess.CalledProcessError as e:
        print(f"Error al ejecutar el comando netsh: {e}\nSalida del comando:
{e.output.decode('cp850')}")
        return None
    except FileNotFoundError:
        print("El comando netsh no se encontró. Asegúrate de que la ruta sea correcta.")
        return None
    except UnicodeDecodeError as e:
        print(f"Error de decodificación: {e}")
        return None
    return resultado
```

Figura 27 Configuración Inicial del Redes Cercanas [15]

El proceso que ocurre, está directamente relacionado con (figura del comando Nesh), ya que utiliza un patrón de búsqueda para encontrar los detalles, nombre de red (SSID), el canal y la fuerza en la señal. En base a esta información, la organiza en un diccionario, y si el nombre de la red este vacío, lo reemplaza por SSID Sin Nombre. Finaliza con la organización de todos los datos de las redes encontradas.

```
def procesar_salida_netsh(salida):
    redes = []
    patron = r"SSID\s+\d+\s+:\s*(.*?)\s+.*?Canal\s+:\s*(\d+)\s+.*?Señal\s+:\s*(\d+)%"
    matches = re.findall(patron, salida, re.DOTALL)
    for ssid, canal, senal in matches:
        redes.append({
            "ssid": ssid.strip() if ssid.strip() else "SSID Sin Nombre",
            "canal": int(canal),
            "senal": int(senal)
        })
    return redes
```

Figura 28 Información Detallada de las Redes Cercanas [14]

En esta función del algoritmo se obtiene la lista de redes cercanas (ejecutar_comando_netsh ()). Se ejecuta el comando y recopila las redes cercanas con la función (procesar_salida_netsh ()). Por último, filtra las redes para quedarse solo con las que están en la banda 2.4 GHz.

```
def obtener_redes_vecinas():
    salida_netsh = ejecutar_comando_netsh()
    if salida_netsh:
        redes = procesar_salida_netsh(salida_netsh)
        redes = [r for r in redes if 1 <= r["canal"] <= 11]
        return redes
    return []
```

Figura 29 Filtro para obtener las Redes en la Banda 2.4 GHz[16]

En esta parte del algoritmo, tomamos el canal individual de AP y obtiene las rede cercanas utilizando (obtener_redes_vecinas ()). Continúa con una comparación entre el canal individual y los canales de las redes vecinas, clasificándola. Si la señal de una red es muy débil (-30%), la ignora. Procede a establecer parámetros para generación de interferencias entre los canales, si está en el mismo canal, cuenta con un solapamiento del 100%, y si se encuentra en canales cercanos la interferencia es menor, y así sucesivamente. Finalmente, devuelve el valor de la interferencia en negativo, ya que la finalidad es buscar el canal con menor interferencia.

```
def evaluar_interferencia(configuracion_ap):
    mi_canal = configuracion_ap[0]
    redes_vecinas = obtener_redes_vecinas()
    print(f"Redes vecinas utilizadas en el cálculo: {redes_vecinas}")
    interferencia = 0

    if redes_vecinas:
        for red in redes_vecinas:
            canal_vecino = red["canal"]
            fuerza_vecino = red["senal"]

            if fuerza_vecino < 30:
                continue

            distancia = abs(mi_canal - canal_vecino)
            if distancia == 0:
                solapamiento = 100
            elif distancia == 1:
                solapamiento = 50
            elif distancia == 2:
                solapamiento = 25
            else:
                solapamiento = 0

            ocupacion = red.get("ocupacion", 0)
            interferencia += (solapamiento * fuerza_vecino * (1 + ocupacion / 100)) / 100
        print(f"Interferencia calculada para canal {mi_canal}: {interferencia}")
    else:
        interferencia = 0

    return -interferencia,
```

Figura 30 Comparativa entre los Individuos y los Canales Vecinos [7]

Continuamos con el proceso de mutación en los canales, en base a las validaciones previas de tamaño_configuraciones. Cada individuo (punto de acceso Wi-Fi con un canal asignado), cuenta con una configuración aleatoria que se compara en cuanto a la cantidad de interferencia y solapamiento causada con las redes vecinas. En base a los resultados en las configuraciones, determina la probabilidad de mutación que es del 0.1, lo que significa que hay una probabilidad del 10% en que el canal de cada configuración (en nuestro caso, 20 simulación de redes), se cambie aleatoriamente en base al valor arrojado por la comparativa entre las redes, con la probabilidad establecida, si el número es menor a (probabilidad_cambio_canal) es posible se realice el cambio de canal. Si la mutación en la configuración (cambio de canal), mejora la presencia de interferencias en comparación con las redes vecinas se mantiene el cambio, si aumenta se revierte el cambio. Este proceso se repite en cada una de las 50 generaciones establecidas, permitiendo una evolución a lo largo del tiempo.

```
def cambiar_canal(configuracion_ap, canales):
    if random.random( ) < probabilidad_cambio_canal:
        nuevo_canal = random.choice(canales)
        while nuevo_canal == configuracion_ap[0]:
            nuevo_canal = random.choice(canales)
        configuracion_ap[0] = nuevo_canal
    return configuracion_ap,
```

Figura 31 Mutación en los individuos [17]

Después de la explicación de cada línea del algoritmo genético, se guarda el script en formato .py y se guarda en la carpeta de scripts del proyecto, en este caso se deja con nombre de wifi_vecinos.

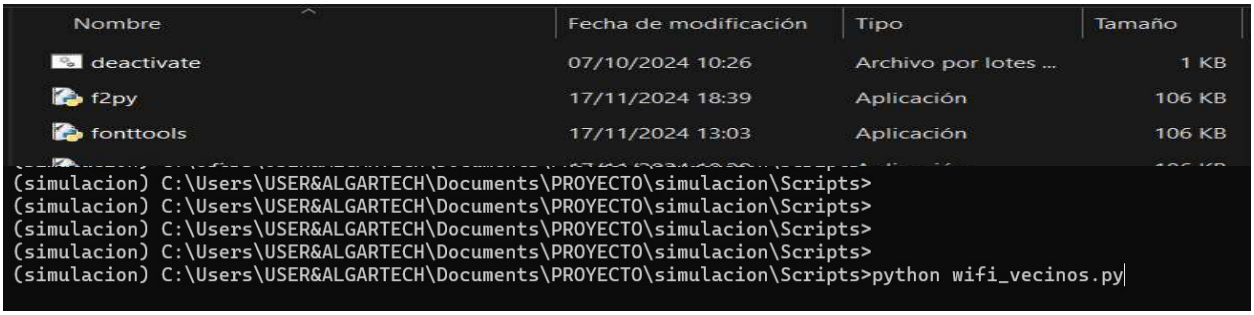


Figura 33 Funcionamiento del Script - Creación Propia

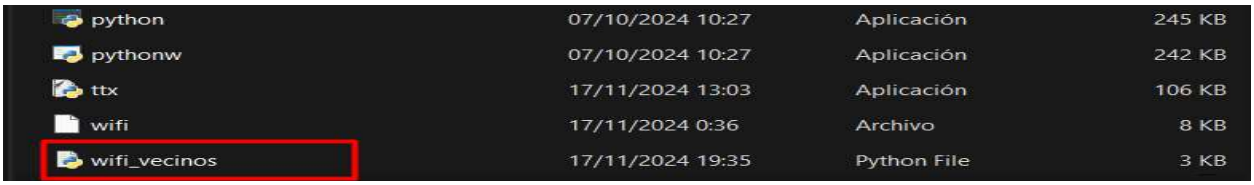


Figura 32 Generación y Almacenamiento de Script - Creación Propia

Y con el archivo .py guardado, vamos a CMD nuevamente para correr el script, ingresamos hasta la carpeta scripts e ingresamos el comando Python wifi_vecinos.py para correr el script.

Como resultado final, nos arroja el mejor canal de acuerdo a la generación del algoritmo genético. Permitiendo conocer y complementar las herramientas tradiciones con un análisis detallado, que en base a parámetros establecidos nos da como resultado una implementación exitosa.

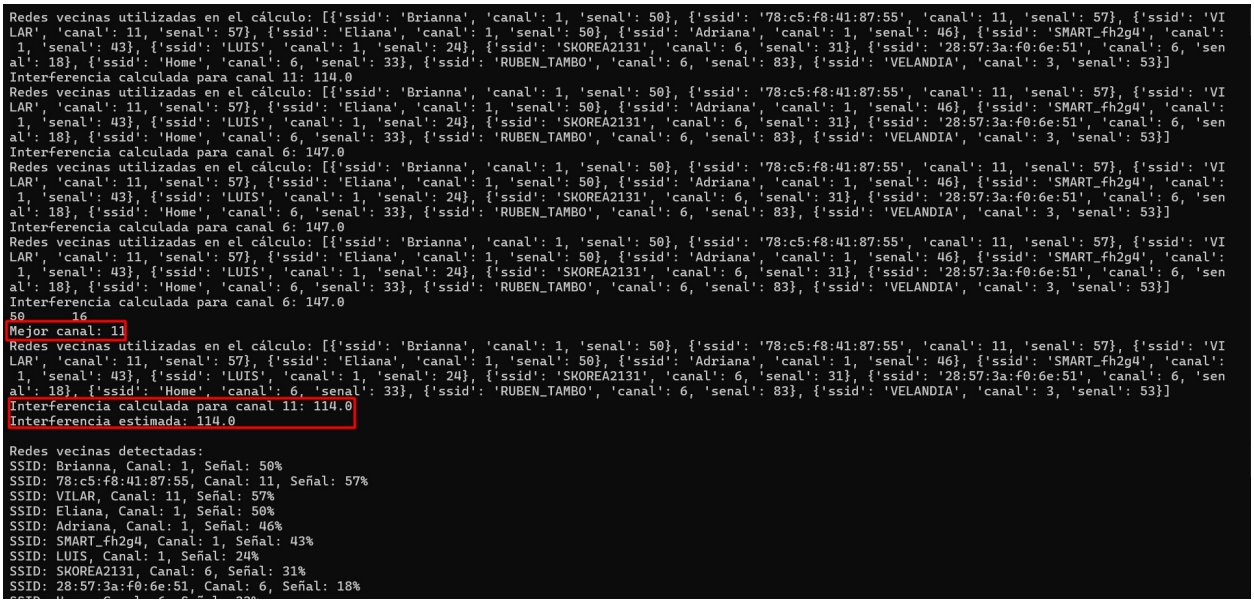


Figura 34 Resultado Final, Canal Idóneo - Creación Propia

10 Conclusiones

Abordamos de manera eficiente la problemática planteada de las interferencias en las redes Wi-Fi 2.4 GHz en entornos domésticos. Realizamos la simulación en un entorno virtualizado, el análisis en un entorno real y la implementación de un algoritmo genético, pudimos demostrar que los métodos tradicionales, como el cambio de canal de manera manual y la optimización de los nodos con la implementación realizada, son estrategias que nos generar una mejora significativa en la calidad de la señal y estabilidad en la conexión.
Frente a la simulación, obtuvimos conocimiento en las herramientas, pensamiento lógico y generación de entornos que nos permitieran obtener los resultados deseados, y lo más importante para nosotros encontrar un camino a seguir en el proyecto de grado, logramos generar una implementación funcional y con sentido.

Nuestros resultados confirmaron que la congestión y la superposición presente en los canales en la banda 2.4 GHz son factores que realmente afectan de manera significativa el rendimiento de las redes Wi-Fi domésticas. Evidenciamos que en el cambio del canal se obtuvo una mejora significativa en la descarga y carga, mejorando la calidad de la conexión. Sin embargo, es necesario una búsqueda y análisis previo para el funcionamiento de la herramienta que pueda permitir esta mejora en la red del usuario. Frente a esa problemática continuamos con una implementación que nos permitió complementar los métodos tradicionales con el análisis y designación del mejor canal para el nodo.

A futuro se precisa la generación de un algoritmo genético completamente automatizado con la base expuesta en este proyecto de grado, permitiendo con un solo clic el análisis y asignación del canal más adecuado para cada nodo que se encuentre en el entorno doméstico en la banda 2.4 GHz.

11 Bibliografías

[1] M. Bullmann, T. Fetzer, F. Ebner, M. Ebner, F. Deinzer, y M. Grzegorzek, “Comparación de Métodos de Posicionamiento Interior WiFi FTM-Basado en RSSI de 2,4 GHz en Escenarios Realistas”, *Sensors (Switzerland)*, vol. 20, núm. 16, pp. 1–26, ago. 2020, doi: 10.3390/s20164515.

[2] I. Tecnológico, D. E. Acapulco, Q. Para, O. El, y T. De, “ESTUDIO DE LOS ESQUEMAS DE MODULACIÓN Y CODIFICACIÓN DE CANAL APLICADOS EN EL ESTÁNDAR IEEE 802.11n UTILIZANDO EL SIMULADOR NS-3 OPCIÓN I: TESIS PROFESIONAL”, dic. 2018.

[3] Cees Links, “The Evolution of Wi-Fi networks: from IEEE 802.11 to Wi-Fi 6E”. Consultado: el 17 de octubre de 2024. [En línea]. Disponible en: https://www.wevolver.com/article/the-evolution-of-wi-fi-networks-from-ieee-80211-to-wi-fi-6e

[4] Rajesh K Mishra, Yitao Chen, y Ian P. Roberts, *Detección de Colisiones en Redes Wi-Fi Densas utilizando Cancelación de Autointerferencia*. Texas: IEEE, 2020.

[5] B. R. Tamma, T. V. Pasca, y A. M. Baswade, “Sobre la convergencia y coexistencia de redes LTE y Wi-Fi”, *CSI Transactions on ICT*, vol. 7, núm. 3, pp. 1–6, sep. 2020, doi: 10.1007/s40012-019-00231-0.

[6] ALBERTO MOLINA CUELLO, “APRENDIZAJE AUTOMÁTICO CON ALGORITMOS GENÉTICOS Y REDES NEURONALES”, jun. 2020.

[7] “Algoritmo Genético en Python: Ejemplos y Código para su Implementación”. Consultado: el 25 de octubre de 2024. [En línea]. Disponible en: https://conectandoideas.net/algoritmo-genetico-python-ejemplo/

[8] A. de la Peña y P. Truyol, “ALGORITMOS GENÉTICOS”.

[9] X. Cui, J. Yang, J. Li, y C. Wu, “Algoritmo Genético Mejorado para Optimizar la Posicionamiento en Interiores de Wi-Fi Basado en una Red Neuronal Artificial”, *IEEE Access*, vol. 8, pp. 1–8, 2020, doi: 10.1109/ACCESS.2020.2988322.

[10] C. Esto, C. Xuerong, Y. Jin, J. Li, y W. Chunlei, “Algoritmo Genético Mejorado para Optimizar el Posicionamiento Interior Basado en Red Neural Artificial”, pp. 1–18, 2015, doi: 10.1109/ACCESO.2020.2988322.

[11] J. Wang, X. Fang, D. Niyato, y T. Liu, “Next-Generation Wi-Fi Networks with Generative AI: Design and Insights”, ago. 2024.

[12] JUAREZ ASTUDILLO FRANCHESCO DAVID, “PROPUESTA DE IMPLEMENTACIÓN DE RED ESTRUCTURADA ADMINISTRATIVA CON UBUNTU”, Perú, 2020.

[13] VICTOR HUGO GONZÁLEZ ESPINOZA y ALEX ENRIQUE MOROCHO LOVATO, “SIMULACIÓN Y ANÁLISIS DEL RENDIMIENTO DE UNA RED LTE MEDIANTE EL SOFTWARE NS3 BAJO CONDICIONES DE TRÁFICO MULTIMEDIA”, Quito, nov. 2017.

[14] “Operators and Algorithms — DEAP 1.4.1 documentation”. Consultado: el 24 de octubre de 2024. [En línea]. Disponible en: https://deap.readthedocs.io/en/master/tutorials/basic/part2.html#crossover

[15] “CalledProcessError for Subprocess.check_output for wlan in windows”. Consultado: el 24 de noviembre de 2024. [En línea]. Disponible en: https://stackoverflow.com/questions/63435708/calledprocesserror-for-subprocess-check-output-for-wlan-in-windows

[16] “GitHub - scivision/scan-wifi-python: CoreLocation, netsh, nmcli: scan Wifi from Python on Windows, MacOS, Linux”. Consultado: el 25 de octubre de 2024. [En línea]. Disponible en: https://github.com/scivision/scan-wifi-python

[17] “DEAP documentation — DEAP 1.4.1 documentation”. Consultado: el 26 de octubre de 2024. [En línea]. Disponible en: https://deap.readthedocs.io/en/master/