UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

TÍTULO DEL TRABAJO DE TITULACIÓN

SIMULACIÓN Y ANÁLISIS DE CALIDAD DE SERVICIO (QOS) EN REDES INALÁMBRICAS MEDIANTE PHYTON

AUTOR

TUNATO POZO JAVIER ALEJANDRO

MODALIDAD EXAMEN COMPLEXIVO

Previo a la obtención del grado académico en
INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

TUTOR

ING. SHENDRY BALMORE ROSERO VÁZQUEZ, MSC

Santa Elena, Ecuador

Año 2024

UPSE

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

TRIBUNAL DE SUSTENTACIÓN

Ing. José Sánchez Aquino, Mgt.
DIRECTOR DE LA CARRERA

Ing. Shendry Rosero Vázquez, Msc.
TUTOR

Ing. Carlos Castillo Yagual, Mgt.
DOCENTE ESPECIALISTA

Ing. Marjorie Coronel Suárez, Mgt.
DOCENTE GUÍA UIC

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que luego de haber dirigido científica y técnicamente el desarrollo y estructura final del trabajo, este cumple y se ajusta a los estándares académicos, razón por el cual apruebo en todas sus partes el presente trabajo de titulación que fue realizado en su totalidad por Tunato Pozo Javier Alejandro, como requerimiento para la obtención del título de Ingeniero en Tecnologías de la Información.

La Libertad, a los 12 días del mes de diciembre del año 2024

TUTOR

Shendry Rosero

Firmado digitalmente por Shendry Rosero
DN: cn=Shendry Rosero gn=Shendry Rosero c=ES Spain l=ES Spain e=shendry.rosero.v@gmail.com
Motivo: Soy el autor de este documento
Ubicación:
Fecha: 2024-12-11 17:24-05:00

Ing. Shendry Rosero Vázquez, Msc.

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

Yo, **TUNATO POZO JAVIER ALEJANDRO**

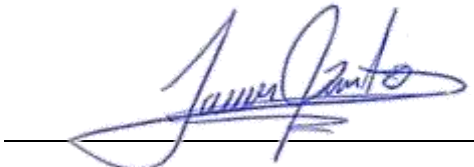
DECLARO QUE:

El trabajo de Titulación, **Simulación y Análisis de calidad de servicio (QoS) en redes inalámbricas mediante Phyton** previo a la obtención del título en Ingeniero en Tecnologías de la Información, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Titulación referido.

La Libertad, a los 12 días del mes de diciembre del año 2024

EL AUTOR



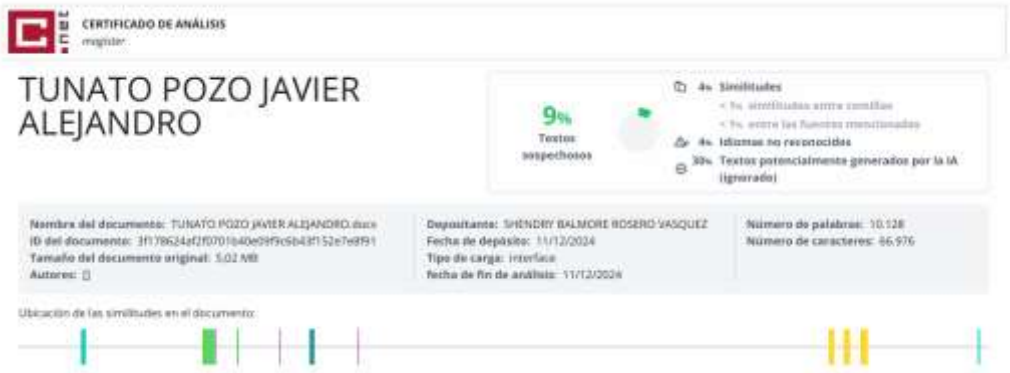
Tunato Pozo Javier Alejandro

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CERTIFICACIÓN DE ANTIPLAGIO

Certifico que después de revisar el documento final del trabajo de titulación denominado **Simulación y Análisis de calidad de servicio (QoS) en redes inalámbricas mediante Phyton**, presentado por el estudiante, Tunato Pozo Javier Alejandro fue enviado al Sistema Antiplagio, presentando un porcentaje de similitud correspondiente al 09%, por lo que se aprueba el trabajo para que continúe con el proceso de titulación.


CERTIFICADO DE ANÁLISIS

TUNATO POZO JAVIER ALEJANDRO

9% Textos sospechosos

4% Similitudes

4% Idiomas no reconocidos

30% Textos potencialmente generados por la IA (ignorado)

Nombre del documento: TUNATO POZO JAVIER ALEJANDRO.docx
ID del documento: 3f178624af2f0701b40e09f9c6b43f152e7e8f91
Tamaño del documento original: 5.02 MB
Autores: []

Depositante: SHENDRY BALMORE ROSERO VASQUEZ
Fecha de depósito: 11/12/2024
Tipo de carga: interfaz
Fecha de fin de análisis: 11/12/2024

Número de palabras: 10.128
Número de caracteres: 66.976

Ubicación de las similitudes en el documento:

TUTOR

Shendry Rosero

Firmado digitalmente por Shendry Rosero
DN: cn=Shendry Rosero gn=Shendry Rosero c=ES Spain l=ES Spain e=shendry.rosero.x@gmail.com
Motivo: Soy el autor de este documento
Ubicación:
Fecha: 2024-12-11 17:24:06:00

Ing. Shendry Rosero Vázquez, Msc.

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

AUTORIZACIÓN

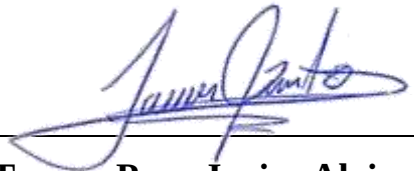
Yo, **TUNATO POZO JAVIER ALEJANDRO**

Autorizo a la Universidad Estatal Península de Santa Elena, para que haga de este trabajo de titulación o parte de él, un documento disponible para su lectura consulta y procesos de investigación, según las normas de la Institución.

Cedo los derechos en línea patrimoniales del trabajo de titulación con fines de difusión pública, dentro de las regulaciones de la Universidad, siempre y cuando esta reproducción no suponga una ganancia económica y se realice respetando mis derechos de autor

La Libertad, a los 12 días del mes de diciembre del año 2024

EL AUTOR



Tunato Pozo Javier Alejandro

AGRADECIMIENTO

A lo largo de este camino he experimentado momentos que han puesto a prueba mi determinación y resistencia. Hoy, al concluir este importante capítulo de mi vida académica, quiero expresar mi más profunda gratitud a quienes han sido pilares fundamentales en este proceso.

Primeramente, a mis padres, quienes han sido mi sostén incondicional. Su apoyo emocional y económico ha sido el cimiento que me ha permitido seguir adelante incluso en los momentos más difíciles. Gracias por creer en mí, por su paciencia y por ser mi refugio en los momentos de adversidad.

A mi esposa, quien ha sido testigo de cada desafío y ha estado a mi lado con amor, comprensión y fortaleza. Tu apoyo ha sido mi mayor motivación, tu paciencia mi consuelo, y tu amor mi energía para continuar.

Un agradecimiento especial a mi tutor que me ha acompañado en este proceso. Su guía profesional, sus conocimientos y su dedicación han sido fundamentales para superar los obstáculos inesperados que se presentaron durante la elaboración de esta tesis.

A mis amigos de universidad, gracias por su apoyo, por los momentos compartidos y por las conversaciones que me ayudaron a crecer.

Esta tesis no es solo el resultado de mi esfuerzo individual, sino el fruto del amor, la paciencia y el apoyo de todas estas personas maravillosas que han estado a mi lado.

Javier Alejandro Tunato Pozo

DEDICATORIA

A mis padres, pilares fundamentales de mi vida, quienes, con su amor incondicional, sacrificio y dedicación me enseñaron el valor del esfuerzo y la perseverancia. Su apoyo constante ha sido mi más grande fortaleza en los momentos de duda y desafío. A mis queridas hermanas, compañeras de vida, confidentes y aliadas, que siempre han estado a mi lado, celebrando mis triunfos y sosteniéndome en los momentos más difíciles.

A mi amada esposa, quien ha sido testigo silencioso de cada desvelo, de cada esfuerzo y de cada sacrificio. Tu amor, comprensión y apoyo incondicional han sido el motor que me impulsa a seguir adelante. A mi adorada hija Julieth, luz de mis días, inspiración constante y razón de ser, que con su sonrisa y ternura me recuerda que cada esfuerzo vale la pena.

A mi querido abuelito, que partió al cielo cuando apenas comenzaba este viaje académico, sé que desde donde estés, me acompañas y me cuidas. Tu memoria es un abrazo eterno que me impulsa a ser mejor cada día. Tus enseñanzas y tu amor incondicional siguen siendo mi guía, y cada logro lleva impresa tu huella imborrable de amor y esperanza.

Este logro no es solo mío, es el resultado de un camino recorrido con amor, sacrificio y apoyo. Gracias por haber creído en mí, incluso cuando yo mismo dudaba, y que han sido mi sostén en cada etapa de este hermoso viaje académico y personal.

Javier Alejandro Tunato Pozo

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El siguiente trabajo aborda un análisis del impacto que tiene la calidad de servicio (QoS) en redes mediante una herramienta de simulación realizada en Python con la finalidad de evaluar su rendimiento en términos de latencia, perdida de paquetes y uso de ancho de banda. La metodología realizada incluye simulaciones controladas donde se emplean bibliotecas somo Simpy, Scapy y Matplotlib. Los resultados nos da a entender que ciertas políticas de calidad de servicio WFQ y RED que permiten mejorar significativamente el desempeño de la red, reduciendo la congestión y optimizando el ancho de banda, a comparación de FIFO siendo esencial la implementación de la calidad de servicio (QoS) para garantizar el rendimiento en escenarios de alta congestión de paquetes, dándole una utilidad a la herramienta como una plataforma accesible para el análisis del comportamiento de las políticas (QoS) en la red.

Palabras claves: Calidad de Servicio (QoS), redes, simulación, Python, políticas de QoS.

ABSTRACT

This study analyzes the impact of Quality of Service (QoS) in networks using a Python-based simulation tool to evaluate performance in terms of latency, packet loss, and bandwidth usage. The methodology includes controlled simulations leveraging libraries such as Simpy, Scapy, and Matplotlib. The results demonstrate that specific QoS policies like Weighted Fair Queuing (WFQ) and Random Early Detection (RED) significantly enhance network performance by reducing congestion and optimizing bandwidth compared to First In, First Out (FIFO). This highlights the essential role of QoS implementation in ensuring performance in high-congestion scenarios and positions the developed tool as an accessible platform for analyzing QoS policy behavior in networks.

Keywords: Quality of Service (QoS), networks, simulation, Python, QoS policies.

INTRODUCCIÓN

El rápido crecimiento del tráfico de red y los servicios en línea han creado la necesidad de implementar mecanismos de gestión de tráfico efectivos en sistemas de comunicación de red. Se menciona que:

A medida que aumenta la congestión, los flujos de paquetes entre los emisores y los destinos sufrirán aumentos en el retardo y, para alta congestión, pérdidas de paquetes [1]. Para hacer frente a este desafío, los proveedores han incorporado mecanismos de calidad de servicio en routers modernos. Dichos mecanismos permiten diferenciar entre tipos de tráfico, como video, datos y voz, al hacer que los paquetes de priorización se procesen de manera más eficiente, lo que aumenta la eficiencia y reduce la congestión de la red. Además, se destaca que:

El control de congestión es responsabilidad conjunta de la red y de los usuarios finales. La red (esto es, el conjunto de gestores de tramas) es el mejor lugar para llevar a cabo la supervisión del grado de congestión, mientras que los usuarios finales constituyen el mejor punto para el control de la congestión mediante la limitación del flujo de tráfico [1].

La implementación de políticas de Calidad de Servicio (QoS) sigue siendo en gran medida un área de investigación no explorada. Si bien la teoría está bien documentada, la falta de estudios en situaciones de la vida real en entornos con niveles de alto tráfico esto mantiene la incertidumbre con respecto a cómo afectan estos mecanismos y analizar si un router hace un mejor trabajo manejando paquetes. Además, actualmente no existen herramientas accesibles y flexibles que permitan a los investigadores simular y analizar el comportamiento de los routers en escenarios realistas con diferentes configuraciones de políticas de Calidad de Servicio (QoS). Si bien hay disponible software de simulación de redes para partículas masivas, muchos de ellos son costosos, difíciles de usar o simplemente no permiten suficiente personalización para evaluar escenarios realistas. Este trabajo busca aportar al desarrollo de una herramienta que permita la simulación de tal escenario para optimizar configuraciones de políticas de Calidad de Servicio (QoS)en redes modernas para evaluar la latencia, la pérdida de paquetes y la eficiencia en la gestión de ancho de banda, de esta manera ayudándolos a comprender mejor cómo se traduce las políticas de Calidad de Servicio (QoS)en el rendimiento práctico de las redes.

CAPITULO 1. FUNDAMETACION

1.1. ANTECEDENTES

Uno de los problemas prioritarios de las videoconferencias es el retardo en la recepción de imagen y sonido. Por ejemplo, al enviar un mail, un retardo de 30 segundos es imperceptible; sin embargo, en una aplicación de tiempo real, un retraso de tan solo 5 segundos se considera un fracaso [2]. Un pequeño retardo puede afectar gravemente la experiencia del usuario. En una videoconferencia, un retraso de 5 segundos en la transmisión de audio o video puede interrumpir la comunicación de manera significativa, generando frustración.

El tráfico en las redes de datos es en ráfagas. Normalmente llega a velocidades no uniformes a medida que la velocidad del tráfico varía (por ejemplo, videoconferencias con compresión), los usuarios interactúan con las aplicaciones (por ejemplo, al navegar por una nueva página web) y las computadoras cambian de tarea [3]. Las ráfagas de tráfico son más difíciles de gestionar que el tráfico de velocidad constante porque pueden llenar los búferes y hacer que se pierdan paquetes [3]. El tráfico en ráfagas, como se menciona en el párrafo citado, es un comportamiento común en las redes de datos debido a la variedad de aplicaciones utilizadas por los usuarios, la variabilidad del tráfico puede generar problemas en la red, especialmente en momentos de alta demanda, causando congestión y latencia.

En las Wireless LAN también se presentan problemas tales como la disponibilidad de ancho de banda limitada, por lo que resulta fundamental poder dotarlas de características de Calidad de Servicio (QoS). [4] Los mecanismos de Calidad de Servicio (QoS) han sido incorporados en routers para priorizar ciertos tipos de tráfico (voz o video) sobre otras (correo electrónico). Existen herramientas de simulación de redes, pero muchas de ellas son costosas, difíciles de usar o no permiten evaluar escenarios realistas.

1.2. DESCRIPCIÓN DEL PROYECTO

Este proyecto tiene como finalidad desarrollar una herramienta, codificada en Python, para simular y analizar el comportamiento de los routers cuando se aplican diferentes mecanismos de Calidad de Servicio (QoS). Esto nos permitirá evaluar el rendimiento de la red o analizar parámetros como latencia, la pérdida de paquetes y el uso del ancho de banda al aplicar políticas de Calidad de Servicio (QoS).

La herramienta estará enfocada en dos principales tareas. En la simulación del tráfico de red se generarán distintos tipos de tráfico, como voz, video y datos, cada uno con características únicas de prioridad y ancho de banda para simular diferentes situaciones comunes en redes reales, como sobrecarga de tráfico o uso equilibrado. Por otro lado, la configuración de mecanismos de Calidad de Servicio (QoS) donde incluiremos la gestión de colas, priorización de paquetes y asignación de ancho de banda a flujos de tráfico específicos para observar su impacto en el rendimiento de la red.

El desarrollo de la herramienta será completamente codificado en Python, utilizando bibliotecas como **Scapy** para la manipulación de tráfico de red, **SimPy** para la simulación de eventos en el tiempo, y **Matplotlib** o **Plotly** para visualizar los resultados. Estas bibliotecas permiten crear un entorno flexible y fácil de usar para que los usuarios puedan diseñar sus propios flujos de tráfico y aplicar diferentes configuraciones de Calidad de Servicio (QoS).

1.3. OBJETIVOS DEL PROYECTO

1.3.1. OBJETIVO GENERAL

- Desarrollar una herramienta codificada en Python para la simulación enfocada en el análisis del comportamiento de los routers mediante mecanismos de QoS bajo diferentes condiciones de tráfico.

1.3.2. OBJETIVOS ESPECIFICOS

- Analizar las principales políticas de QoS utilizadas en routers, evaluando sus características y aplicaciones prácticas.
- Configurar diferentes políticas de QoS en entornos de red simulados.
- Comparar el rendimiento de la red con y sin la implementación de mecanismos QoS en términos de latencia, pérdida de paquetes y congestión.
- Evaluar los resultados obtenidos de las simulaciones para ofrecer recomendaciones sobre las mejores prácticas en la configuración de QoS para optimizar el rendimiento de redes.

1.4. JUSTIFICACIÓN DEL PROYECTO

En la actualidad se utiliza ampliamente la expresión Calidad de Servicio (QoS, Quality of Service), no sólo en el ámbito de las telecomunicaciones, del cual proviene, sino también en los servicios de banda ancha, inalámbricos y multimedios, que usan el Protocolo Internet (IP, Internet protocol). En las redes y sistemas que se vienen diseñando se tiene en cuenta más frecuentemente la calidad de funcionamiento de extremo a extremo requerida por las aplicaciones de usuario. [5] Este concepto ahora es fundamental para asegurar que las aplicaciones de usuario cumplan con un rendimiento de extremo a extremo adecuado. La Calidad de Servicio (QoS) ahora se usa en muchos otros tipos de redes, como internet y conexiones inalámbricas donde nos debe garantizar que todo funcione sin problemas, evitando retrasos o interrupciones.

Es especialmente importante que durante los periodos de congestión los flujos de tráfico con distintos requisitos sean tratados de forma diferente y se les asigne una calidad de servicio (QoS, Quality of Service) diferente. Por ejemplo, un nodo puede transmitir en la misma cola paquetes de alta prioridad con preferencia sobre paquetes con prioridad menor; o un nodo puede mantener diferentes colas con distintos niveles QoS y dar prioridad a los niveles superiores [1]. Cuando una red se congestiona, es esencial priorizar

el tráfico según su importancia para garantizar que los datos más críticos reciban un mejor trato. Esto permite que las aplicaciones que requieren un rendimiento más alto, como las videollamadas o los servicios en tiempo real, no se vean afectadas por retrasos o pérdidas de información, mientras que los datos menos urgentes pueden esperar.

1.5. ALCANCE DEL PROYECTO

El proyecto abarcará el desarrollo de una herramienta de simulación que permitirá analizar el comportamiento de redes mediante la implementación de políticas de Calidad de Servicio (QoS). El alcance se centrará en:

1. **Simulación de tráfico de red:** La herramienta generará diferentes tipos de tráfico de datos, bajo condiciones variadas para replicar escenarios comunes en redes reales.
2. **Implementación de políticas de Calidad de Servicio (QoS):** Se integrarán políticas de QoS (WFQ, FIFO, RED), permitiendo evaluar el impacto sobre el rendimiento de la red.
3. **Análisis de métricas clave:** La herramienta medirá y analizará métricas como latencia, pérdida de paquetes y utilización del ancho de banda para verificar si las políticas de Calidad de Servicio (QoS) aplicadas generan mejoras en el rendimiento.
4. **Escenarios de congestión de red:** El proyecto también abarcará la simulación de escenarios con diferentes niveles de congestión para evaluar cómo la Calidad de Servicio (QoS) responden en condiciones de alta demanda de recursos.

Sin embargo, no se realizará simulación de redes físicas, ni la interacción directa con routers. La herramienta estará enfocada únicamente en entornos simulados, donde se realizarán pruebas controladas sobre el impacto de Calidad de Servicio (QoS) permitiéndonos explorar y analizar el comportamiento y rendimiento de redes al implementar políticas de Calidad de Servicio (QoS). El proyecto está diseñado para funcionar como una plataforma accesible para el análisis de redes bajo condiciones simuladas, facilitando la investigación en políticas de Calidad de Servicio (QoS) y proporcionando resultados sobre el comportamiento del tráfico en diferentes escenarios.

CAPÍTULO 2. MARCO TEÓRICO Y METODOLOGÍA DE DESARROLLO

2.1. MARCO CONCEPTUAL

Redes de comunicación

Las redes de comunicación son sistemas que permiten la transmisión de datos entre múltiples dispositivos. La estructura y eficiencia de estas redes determinan la calidad del servicio. Una red puede estar compuesta por diferentes elementos como routers, switches, y puntos de acceso, que gestionan el flujo de información [3].

Red inalámbrica

Una red inalámbrica (Wireless Network), son aquellas redes que no tiene cables y se comunicar por "medios no guiados". La transmisión y su recepción se realiza mediante una antena inalámbrica. Dando así que el emisor posee una antena, pero también puede tener varias, además que constan de sistemas que suele tener dos, tres hasta cuatro antemas. La mayor parte de las antenas suelen usarse para varios nodos [6].

Calidad de servicio (QoS)

La calidad del servicio (QoS) se define como la capacidad de una red para proporcionar diversos niveles de servicio a los diferentes tipos de tráfico. Al contar con QoS es posible asegurar una correcta entrega de la información, dando preferencia a aplicaciones de desempeño crítico, donde se comparten simultáneamente los recursos de red con otras aplicaciones no críticas. QoS hace la diferencia al proveer un uso eficiente de los recursos en caso de presentarse congestión en la red, seleccionando un tráfico especifico de ésta, priorizándolo según su importancia relativa, y utilizando métodos de control y evasión de la congestión para darles un tratamiento preferencial. Implementando QoS en una red, se logra un rendimiento de ésta más predecible y una utilización de ancho de banda más eficiente [7].

Retardo (Delay)

El retardo es el tiempo de retraso en la llegada de los paquetes hasta su destino. Al transmitir paquetes de un punto a otro se genera una variación temporal o retraso de flujos de datos a su destino, hoy en día varios factores influyen en el retardo de un paquete que

atraviesa la red, dependiendo de las aplicaciones que se estén orientando las telecomunicaciones se presentan retardos de enrutamiento (router), retardo en colas, retardo de propagación y retardo de serialización, que se producen en cada nodo y enlaces en la red [8].

Variación del Retardo (Jitter)

La Variación del retardo se produce cuando los paquetes transmitidos en una red no llegan, debido a una distorsión en los tiempos de llegada de los paquetes. En si es el efecto del retardo en la comunicación ya que se producen fluctuación en el canal por la diferencia entre varios retardos de paquetes en un mismo flujo. Una fuente potencial de Jitter es que los paquetes consecutivos de un mismo flujo sigan caminos físicos diferentes [9].

Pérdida de Paquetes

La pérdida de paquetes o Packet Loss indica el número de paquetes perdidos durante la transmisión que normalmente se miden en tanto por ciento, hay tres fuentes de pérdidas de paquetes en una red IP estas son, rotura en un enlace físico que evita la transmisión de un paquete, un paquete corrupto debido al ruido detectado por un sistema de checksum y desbordamiento de las memorias producidas por la congestión de la red [8].

First in – First out (FIFO)

Es la técnica más básica de colas. Una cola FIFO es un buffer sencillo que retiene los paquetes salientes hasta que la interfaz las retire en el mismo orden en el que llegaron al buffer [10].

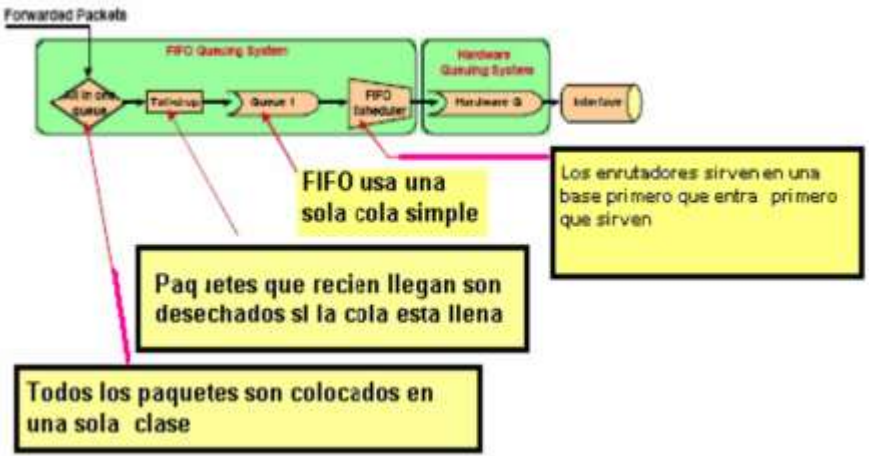


Imagen 1. Encolamiento FIFO. tomado de Diseño e implementacion de QoS.

WQF (Weighted Fair Queuing)

Es un sofisticado proceso y estrategia de construcción de colas en un enrutador el cual requiere muy poca configuración porque el proceso detecta dinámicamente los flujos entre aplicaciones y automáticamente asigna y maneja colas separadas para cada flujo. En términos de WFQ esos flujos son conversaciones a nivel de sesión, Una conversación pudiera ser un sesión Telnet , una transferencia de archivos FTP, un video sobre IP, una descarga de una página Web, o cualquier otro flujo TCO o UDP entre un cliente y servidor [10].

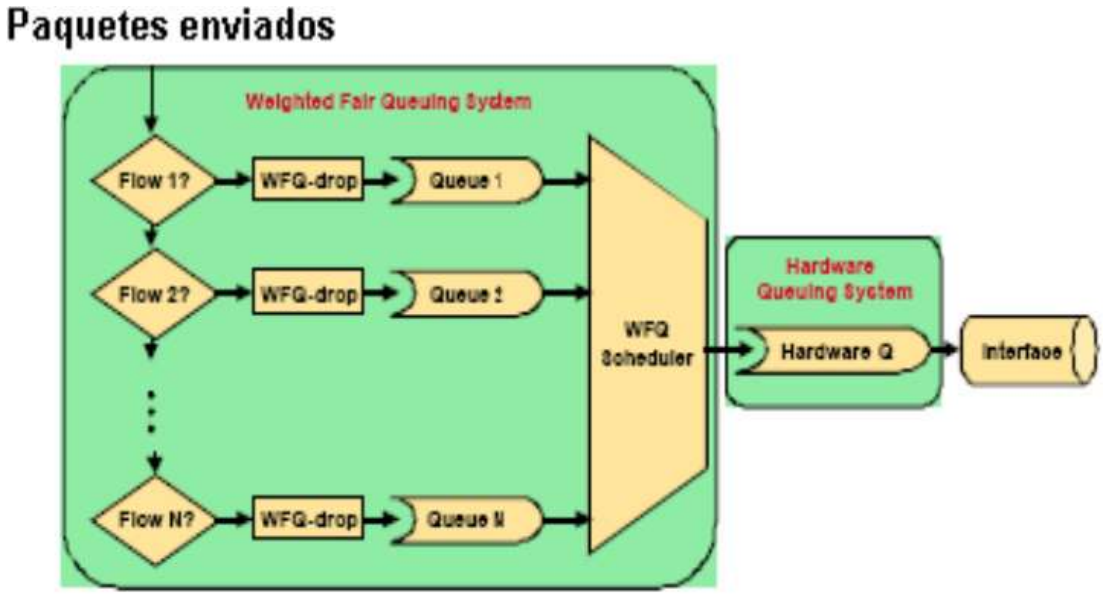


Imagen 2. Encolamiento WFQ. Tomado de Estudio y Configuracion de Calidad de Servicio.

Random Early Detection (RED)

Es una forma de administración activa de la cola. En lugar de esperar pasivamente hasta que los buffers descarten el tráfico, RED descarta aleatoriamente y en forma progresiva paquetes en cuanto detecta una tendencia a la congestión (detección anticipada aleatoria). La probabilidad y velocidad de descarte de los paquetes aumenta con el grado de ocupación de la cola (profundidad de cola). En respuesta a este aumento de descarte de paquetes por parte del receptor, el emisor reduce la tasa de envío hasta tanto se recupera la profundidad de la cola [10].

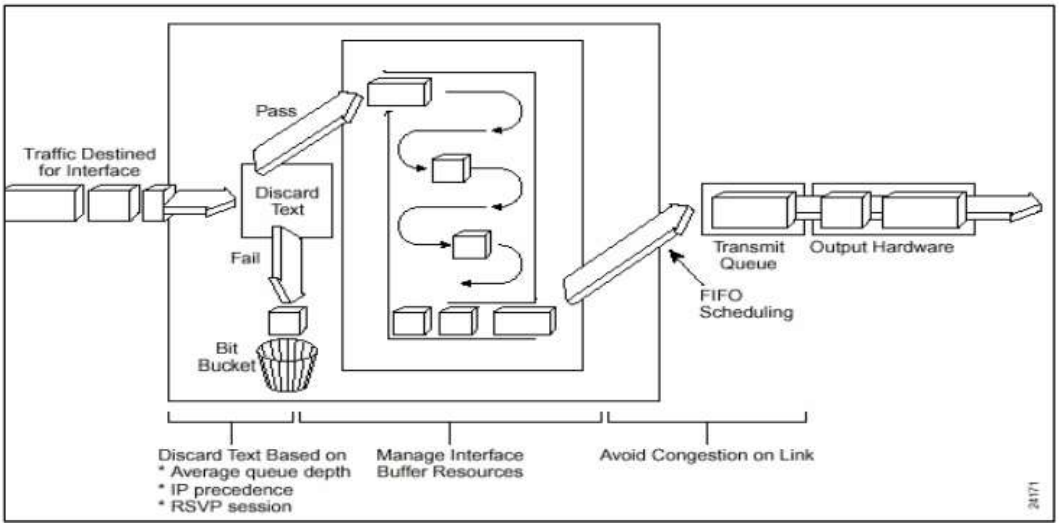


Imagen 3. Encolamiento RED. Tomado de AURA, Ganz, Multimedia Wires Network; Standard and QoS.

Lenguajes de programación

Son sistemas formales diseñados para transmitir instrucciones a una computadora y escribir programas informáticos. Estos lenguajes permiten a los programadores expresar lógica y algoritmos de manera estructurada y comprensible para las máquinas. A través de ellos, se pueden realizar diversas tareas, como cálculos matemáticos, manipulación de datos, interacción con dispositivos y creación de software. Los lenguajes de programación proporcionan un medio eficiente para la comunicación entre humanos y máquinas, permitiendo la implementación de soluciones informáticas [11].

Python

La flexibilidad de Python es un lenguaje flexible que permite la interoperabilidad con múltiples lenguajes, como C/C++ y .NET. Es especialmente adecuado para la construcción de herramientas de inteligencia artificial y aprendizaje automático [12].

Jupyter Notebook

Su característica distintiva radica en la capacidad de ejecutar flujos de trabajo completos de ciencia de datos. Esta herramienta permite llevar a cabo tareas como la limpieza y preparación de datos, la creación y entrenamiento de modelos de aprendizaje automático,

la visualización de datos y muchas otras funcionalidades. Proporciona un entorno integrado que facilita la realización de múltiples etapas del proceso de ciencia de datos de manera eficiente y con un alto grado de automatización [13].

Bibliotecas e herramientas

Las bibliotecas son conjuntos de código predefinido que ofrecen funciones y herramientas específicas para realizar tareas comunes de programación. Están diseñadas para extender la funcionalidad básica del lenguaje y mejorar la eficiencia y conveniencia en la realización de diversas tareas [14].

Asyncio

El módulo asyncio proporciona infraestructura para escribir código simultáneo de un solo subproceso utilizando corrutinas, multiplexando el acceso de E/S a través de sockets y otros recursos, ejecutando clientes y servidores de red y otras primitivas relacionadas [15].

Nest_asyncio

Este módulo parchea asyncio para permitir el uso anidado de asyncio.run y loop.run_until_complete [15].

Random

Este es un módulo que puedes usar para crear números aleatorios [16]**.**

Heapq

Este módulo proporciona una implementación del algoritmo de cola de montón, también conocido como algoritmo de cola de prioridad [17].

Time

Este módulo proporciona varias funciones relacionadas con el tiempo [17].

Datetime

El módulo datetime proporciona clases para manipular fechas y horas [17].

Matplotlib_pyplot

Matplotlib.pyplot es una colección de funciones que hacen que matplotlib funcione como MATLAB. Cada función pyplot realiza algún cambio en una figura: por ejemplo, crea

una figura, crea un área de trazado en una figura, traza algunas líneas en un área de trazado, decora el gráfico con etiquetas, etc [18].

Seaborn

Seaborn es una conocida biblioteca de Python para la visualización de datos, desarrollada sobre Matplotlib, que ofrece una interfaz fácil de usar para producir gráficos estadísticos visualmente atractivos e informativos. Está diseñada para funcionar con marcos de datos de Pandas, lo que facilita la visualización y exploración de datos de forma rápida y eficaz [19].

Numpy

NumPy es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Incorpora una nueva clase de objetos llamados arrays que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación [20].

Pandas

Pandas es una herramienta de análisis y manipulación de datos de código abierto rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación Python [21].

Tkinter

TkInter es la biblioteca gráfica de Python usada para el desarrollo de la interfaz gráfica del proyecto. Esta facilita la construcción y configuración de ventanas con widgets. Estos widgets muestran información al usuario y les permiten interconectar con la aplicación [22].

2.2. MARCO TEÓRICO

Calidad de Servicio (QoS) en Redes Inalámbricas

La demanda de aplicaciones sobre redes que presentan limitaciones de ancho de banda como es el caso de las inalámbricas se impone la necesidad de brindar Calidad de servicio (QoS) en entornos inalámbricos haciendo énfasis al estudio en tecnologías fundamentales como la wifi, 802.11 y la 802.16 y el análisis que describen los parámetros de calidad de servicios, en entornos inalámbricos, así como los métodos que se emplean en las tecnologías mencionadas para obtener dicha Calidad de Servicio [23].

Calidad de servicio (QoS) en Redes WI-FI

En la tecnología WiFi se definió el protocolo IEEE 802.11e para la implementación de QoS el cual agrega un campo al protocolo 802.11 para el control de la QoS, permitiendo diferenciar los tipos de tráficos para darle un trato diferenciado a estos [23].

Los estándares de QoS para redes inalámbricas están definidas como 802.11e y Wi-fi Multimedia (WMN), en este tipo de red es importante analizar como enlazar los campos de prioridad en las cabeceras 802.1p o DSCP con 802.11e. Para mantener QoS de extremo a extremo el controlador inalámbrico y la arquitectura centralizada de AP tienen que llevar a cabo ciertas asociaciones entre las marcas de los datos de tráfico que reciben y el que deben enviar, para proporcionar un campo equivalente tanto de CoS como al DSCP.

La tecnología WiFi y WiMAX permiten implementaciones de QoS, de manera que la tecnología WiMAX se garantiza la QoS a nivel MAC [24].

Modelos QoS

Existen 3 modelos de QoS llamados Best-Effort, IntServ y DiffServ, los cuales serán definidos a continuación:

- Servicios Integrados (IntServ)

  El modelo de Servicios Integrados (RFC 1633), también conocido como IntServ fue desarrollada por la IETF (Internet Engineering Task Force) para garantizar la QoS de extremo a extremo en aplicaciones en tiempo real. IntServ se basa en la reserva de recursos de la red según los requisitos de cada aplicación [25].

  IntServ utiliza el Protocolo de señalización RSVP (Resource Reservation Protocol) para señalar explícitamente las necesidades de QoS del tráfico que

atraviesa la red. En IntServ, la aplicación solicita un tipo específico de servicio de acuerdo con su perfil de tráfico. La aplicación enviará los datos a través de la red solo después de que reciba una confirmación de la red [26].

IntServ reserva los recursos a lo largo de toda la ruta, además requiere que todos los nodos intermedios mantengan el estado del flujo. Esto ocasiona un alto consumo de recursos en los nodos de la red y, por lo tanto, no es escalable [27].

- Servicios Diferenciados (DiffServ)

  El modelo de Servicios Diferenciados (RFC 2475), más comúnmente conocido como DiffServ, surge como una alternativa para abordar los problemas de escalabilidad del modelo IntServ. A diferencia de IntServ, que utiliza un enfoque de extremo a extremo, DiffServ utiliza un enfoque de salto a salto, además utiliza flujos agregados en lugar de flujos individuales [26].

  La escalabilidad se consigue por medio funciones complejas de clasificación y acondicionamiento de tráfico implementadas en los nodos límite de la red. Estos nodos serán los encargados de mapear los paquetes entrantes en alguna de las clases de tráfico soportadas por la red. Para la identificación de los agregados de tráfico se utiliza un código DSCP (DiffServ Code Point). Cada paquete que contiene el mismo código DSCP recibe el mismo trato en cada nodo. PHB (Per Hop Behaviour) define un comportamiento específico para cada clase de tráfico en cada nodo de la red [26].

2.3. METODOLOGÍA DEL PROYECTO

2.3.1. METODOLOGÍA DE INVESTIGACIÓN

La información recogida podrá emplearse en análisis cuantitativos para identificar y conocer la magnitud de los problemas que se suponen o se conocen en forma parcial; también puede utilizarse para un análisis de correlación para probar hipótesis descriptivas [28]. En este trabajo se utilizará una metodología cuantitativa y experimental, una vez que esta investigación se basará en el análisis de datos resultantes de las simulaciones, teniendo en cuenta los siguientes criterios: tipo de investigación; fundamentos teóricos; delimitación del estudio; y criterios de validación *(Tabla. 1)*.

Criterios	Descripción
Tipo de investigación	Investigación experimental aplicada
Fundamentos teóricos	Revisión de literatura sobre políticas de QoS y conceptos de calidad de servicio en redes.
Delimitación del estudio	Simulaciones realizadas en entornos controlados con Python (Jupyter Notebook).
Criterios de validación	Análisis de parámetros, como es el caso de latencia, pérdida de paquetes, y uso de ancho de banda, con el fin de medir la efectividad de las políticas de QoS.

Tabla 1. Criterios de la metodología adoptada.

2.3.2. TÉCNICAS E INSTRUMENTOS DE RECOLECCIÓN DE DATOS

La recolección de datos en este proyecto se realiza mediante una simulación que permite analizar cómo diferentes políticas de Calidad de Servicio (QoS) impactan en el comportamiento de una red.

Se elegirán como técnicas de recopilación de datos las simulaciones por computadora; y las mediciones de parámetros *(Tabla 2).*

Criterios	Descripción
Simulaciones por computadora	Se ejecutarán escenarios de prueba en Jupyter Notebook utilizando bibliotecas de Python como simpy, scapy, asyncio, random, heaqp, matplotlib, tkinter, para la simulación del tráfico de red.
Mediciones de parámetros	Mediante la implementación de scripts para capturar y registrar métricas de latencia, pérdida de paquetes y utilización del ancho de banda. Para registrar y visualizar los resultados se utilizarán herramientas como Panda y Matplotlib.

Tabla 2. Técnicas de recopilación de datos.

2.3.2.1. METODOLOGÍA DE DESARROLLO

Utilizaremos una variante del modelo del ciclo de vida iterativo incremental o prototipado evolutivo. Esta variante es adecuada para el desarrollo de la herramienta con validaciones frecuentes con las siguientes fases: planificación; desarrollo; prueba; análisis y validación; y entrega *(Tabla 4).*

Fases	**Actividades**
Planificación	• Definición de requisitos a nivel de herramientas, incluyendo tipos de políticas de QoS y parámetros de análisis. • Elegir bibliotecas y marcos de Python apropiados.
Desarrollo	• Codificar el simulador en Jupyter Notebook. • Desarrollar funciones para configurar diferentes políticas de QoS y registrar parámetros de rendimiento. • Validar cada módulo individualmente.
Prueba	• Realizar las simulaciones en diferentes condiciones de tráfico. • Comparar los resultados obtenidos con y sin mecanismos QoS.
Análisis y Validación	• Procesar los datos recopilados y crear los gráficos comparativos. • Evaluar la eficacia del simulador a la hora de generar conocimientos relevantes sobre la QoS.
Entrega	• Documentar uso del simulador y recomendaciones para configurar QoS.

Tabla 3. Fases del modelo del ciclo de vida iterativo incremental o prototipado evolutivo.

CAPÍTULO 3. PROPUESTA

3.1. FASE DE PLANIFICACIÓN

3.1.1. REQUERIMIENTOS

3.1.1.1. REQUERIMIENTOS FUNCIONALES

Código	Requerimiento	Descripción del requerimiento
RF-01	Generación de trafico	La herramienta permitirá generar distintos tipos de tráfico con características específicas, como prioridad y tamaño de paquetes.
RF-02	Configuración de QoS	La herramienta permitirá configurar y aplicar políticas de QoS, como FIFO (First In, First Out), WFQ (Weighted Fair Queuing) y RED (Random Early Detection).
RF-03	Medición de Latencia	La herramienta deberá medir y registrar la latencia del tráfico simulado, permitiendo evaluar el retardo causado por cada política de QoS.
RF-04	Medición de perdida de paquetes	Calculará y mostrará el porcentaje de paquetes perdidos bajo diferentes configuraciones de QoS y tipos de tráfico.
RF-05	Uso de ancho de banda	El sistema debe visualizar el uso de ancho de banda de cada tipo de tráfico y cómo afecta la implementación de políticas QoS.

RF-06	Visualización de resultados	La herramienta debe generar gráficos en tiempo real que permitan analizar el rendimiento de la red simulada.
RF-07	Generación de reportes	Al finalizar la simulación, debe crearse un reporte con los resultados clave, incluyendo gráficos de latencia, pérdida de paquetes y uso de ancho de banda.
RF-08	Interfaz de usuario	Proporcionar una interfaz gráfica (GUI) intuitiva para configurar y ejecutar la simulación sin necesidad de conocimientos avanzados de programación.

Tabla 4. Requerimientos funcionales.

3.1.1.2. REQUERIMIENTOS NO FUNCIONALES

Código	Requerimiento	Descripción del requerimiento
RNF-01	Rendimiento	La herramienta debe ser capaz de procesar al menos 1000 paquetes por segundo sin que se vuelva lenta o se trabe, incluso cuando hay mucho tráfico simulado.
RNF-02	Escalabilidad	Debería poder manejar hasta 10,000 paquetes al mismo tiempo para probar mecanismos QoS en situaciones de tráfico intenso sin problemas.

RNF-03	Fácil Uso	La interfaz sencilla e intuitiva, para que cualquier persona pueda configurar y realizar simulaciones.
RNF-04	Portabilidad	La herramienta debería poder instalarse y correr bien en cualquier sistema operativo sin hacer configuraciones complicadas.
RNF-05	Flexibilidad en configuraciones	Debería ser fácil de ajustar configuraciones de mecanismo QoS y el tráfico simulado para adaptarlo a distintos tipos de pruebas.
RNF-06	Visualización clara de gráficos	Los gráficos generados deben ser visualmente claros permitan comprender los resultados sin confusión.
RNF-07	Compatible con versiones de Python	La herramienta debe realizarse y funcionar en versiones de Python 3.8 o superiores, usando librerías conocidas.
RNF-08	Actualización de librerías	Las dependencias y librerías utilizadas deben mantenerse actualizadas, garantizando la seguridad y funcionalidad de la herramienta.

Tabla 5. Requerimientos no funcionales.

3.1.2. SELECCIÓN DE HERRAMIENTAS

Las herramientas seleccionadas nos permiten llevar a cabo las simulaciones y análisis de redes de forma eficiente, asegurándonos resultados confiables. Estas herramientas fueron seleccionadas por su capacidad de adaptabilidad ante necesidades específicas al realizar el análisis de los mecanismos de calidad de servicio (QoS) en redes.

LENGUAJE DE PROGRAMACIÓN

- **Python**

  El desarrollo de la herramienta para la simulación de router y realizar el análisis de la calidad de servicio (QoS) se ha llevado a cabo en Python, siendo este un lenguaje versátil, además su amplia gama de librerías que nos ayudaran a realizar simulaciones de red, análisis de datos y visualización de resultados.

- **Jupyter Notebook**

  Para facilitar los procesos de desarrollo y ejecución de los códigos usamos Jupyter Notebook donde tendremos un ambiente interactivo que nos permite integrar código de manera ágil.

LIBRERÍAS Y FRAMEWORKS

En el proyecto se han utilizado múltiples librerías de Python que nos dan funciones específicas para realizar las simulaciones, análisis y visualización de resultados. Entre las principales tenemos scapy, simpy, matplotlib, etc *(Tabla 6)*.

Librería	**Descripción**
Scapy	Para la generación y manipulación de paquetes de red, lo que permite simular distintos escenarios de tráfico en la red.
Simpy	Para modelar y simular eventos en tiempo discreto, reproduciendo dinámicas de red con políticas de QoS.
Matplotlib y Plotly	Herramientas utilizadas para crear gráficos que representen visualmente las

	métricas analizadas, como latencia y pérdida de paquetes.
Tkinter	Para la realización de la interfaz gráfica, diseñar ventanas o controles (botones, graficos, etc) que permitan interactuar con la simulación.
Pandas	Para almacenar, procesar y analizar las estadísticas de rendimiento del simulador, como latencia, pérdida de paquetes, o uso de ancho de banda.
Numpy	Para cálculos complejos relacionados con las métricas de QoS
Datetime	Para registrar tiempos de la simulación, como cuándo se creó un paquete o cuándo fue procesado.
Json	Para guardar configuraciones del simulador, exportar estadísticas o cargar configuraciones predefinidas de QoS.
Asyncio	Para simular la llegada de paquetes y el procesamiento en tiempo real.
Threading	Para manejar tareas concurrentes como el procesamiento de paquetes en varios routers simultáneamente.
Collections.deque	Maneja colas de datos, representando las colas de salida de los routers y del tráfico.

Time	Para medir los tiempos y permitiéndonos calcular métricas como latencia o intervalos de actualización de estadísticas.
Random	Para realizar procesos aleatorios como la política RED.

Tabla 6. Librerías.

3.1.3. HARDWARE Y RECURSOS COMPUTACIONALES

Para garantizar un desarrollo eficiente y realizar simulaciones de red, se ha establecido un conjunto de recursos informáticos que cumplen con las necesidades específicas del proyecto. Se han seleccionado tomando en cuenta la carga de trabajo asociada con el análisis y la simulación de calidad de servicio (QoS) en redes *(Tabla 7)*.

Elemento	**Característica**	**Descripción**
Computadora	Procesador: Intel Core i7 RAM: 12 GB Almacenamiento: 512 GB	Permite ejecutar simulaciones de manera eficiente, reduciendo tiempos de ejecución y análisis.
Sistema operativo	Windows 10	Compatibilidad con Python y las herramientas necesarias, para el desarrollo de la simulación.
Conexión a internet	Banda ancha de 400Mbps	Es necesaria para descargar dependencias, actualizaciones y realizar investigaciones durante el desarrollo.

Software	Python 3.9	Lenguaje utilizado para realizar la simulación del proyecto.
	Anaconda y Jupyter Notebook	Una interfaz interactiva para desarrollar el código en Python, visualizar resultados y gestionar el entorno virtual.

Tabla 7. Recursos Computacionales.

3.1.4. JUSTIFICACIÓN DE LA PLANIFICACIÓN

Las herramientas y tecnologías elegidas para este proyecto fueron seleccionadas con para cumplir con los objetivos y requerimientos establecidos. Las herramientas no solo son funcionales y confiables, sino también fáciles de usar y adaptar durante el desarrollo de la simulación. El lenguaje de programación y las librerías seleccionadas nos ofrecen la flexibilidad, compatibilidad y adaptabilidad para simular y analizar su rendimiento, mientras que el entorno de desarrollo nos facilita pruebas y la visualización de resultados.

3.2. FASE DE DESARROLLO

3.2.1. ARQUITECTURA DEL SISTEMA

El sistema está diseñado para simular una red virtual que permite gestionar tráfico de datos utilizando políticas de calidad de servicio (QoS), integrando dos componentes principales:

- **Capa de Simulación (Backend):** Contiene la lógica para modelar la red y procesar paquetes de datos.
- **Capa de Presentación (Frontend):** Una interfaz gráfica en Jupyter Notebook que permite interactuar con el sistema de manera visual y dinámica.

3.2.1.1. CAPA DE SIMULACIÓN

Es el componente principal del sistema donde se desarrollará la parte lógica, simulando una red virtual mediante routers, interfaces, paquetes y las políticas de calidad de servicio (QoS).

COMPONENTES PRINCIPALES

3.2.1.1.1. ENUMERACIONES

class QoSPolicy(Enum)

Definimos las políticas de calidad de servicio a utilizar (FIFO, WFQ, RED). En este caso, cada miembro de QoSPolicy tiene un nombre y un valor asociado.

```
8  class QoSPolicy(Enum):
9      FIFO = "FIFO"
0      WFQ = "WFQ"
1      RED = "RED"
```

Imagen 4. Clase QoSPolicy.

class TrafficType(Enum)

Cada miembro de esta enumeración representa un tipo específico de tráfico que podría encontrarse en una red como voz, video, datos y control.

```
3  class TrafficType(Enum):
4      VOICE = "VOICE"
5      VIDEO = "VIDEO"
6      DATA = "DATA"
7      CONTROL = "CONTROL"
```

Imagen 5. Clase TrafficType.

3.2.1.1.2. CLASES

class Packet

Esta clase está diseñada para representar las características esenciales de un paquete para gestionar, analizar o simular flujos de paquetes *(Tabla 8)*.

Atributo	Tipo	Descripción
id	int	Almacena el identificador único del paquete.
source	str	Almacena la dirección de origen del paquete

destination	str	Almacena la dirección de destino del paquete
size	int	Tamaño del paquete
priority	int	Nivel de prioridad del paquete
creation_time	float	Almacena el tiempo de creación del paquete.
traffic_type	str	Tipo de tráfico del paquete

Tabla 8. Atributos clase Packet.

```
19 class Packet:
20     def __init__(self, id: int, source: str, destination: str, size: int,
21                  priority: int, creation_time: float, traffic_type: str):
22         self.id = id
23         self.source = source
24         self.destination = destination
25         self.size = size
26         self.priority = priority
27         self.creation_time = creation_time
28         self.traffic_type = traffic_type
```

Imagen 6. Clase Packet.

class Interface

En esta clase tenemos una capacidad limitada, donde maneja una cola de salida para los paquetes que se procesan.

Atributo	Tipo	Descripción
capacity	int	Define el máximo de paquetes que la interfaz puede manejar.
output_queue	deque	Almacena los paquetes que están en espera para ser procesados.
stats	interfacestats	Monitorea las estadísticas de rendimiento

Tabla 9. Atributos de la clase Interface.

class InterfaceStats

Esta clase lleva un registro de estadísticas relacionadas con el rendimiento de la interfaz. Es decir, rastreara cuántos paquetes se procesan, cuántos se pierden y cuánto tiempo total se ha acumulado en latencia.

```
30 class Interface:
31     def __init__(self, capacity: int):
32         self.capacity = capacity
33         self.output_queue = deque()
34         self.stats = InterfaceStats()
35 
36 class InterfaceStats:
37     def __init__(self):
38         self.packets_processed = 0
39         self.packets_dropped = 0
40         self.total_latency = 0.0
```

Imagen 7. Clase Interface.

class Router

Este código simula el comportamiento de un router que procesa paquetes, toma decisiones sobre la cola de tráfico según las políticas de QoS y mantiene estadísticas de su rendimiento.

- **def __init__**

  Se definen atributos del router *(Tabla 10)*.

Atributo	Descripción
name	Nombre del router
interfaces	Diccionario que mapea nombres de interfaces a objetos Interface.
routing_table	Tabla de enrutamiento que asigna destinos a interfaces.
qos_policy	Política de calidad de servicio.
queues	Colas separadas para diferentes tipos de tráfico (VOZ, VIDEO, DATOS).

traffic_priorities	prioridades del tráfico de paquetes (VOZ = mayor prioridad, DATOS = menor prioridad).
stats	Almacena métricas sobre el tráfico de paquetes (procesados, descartados, latencia, etc.).
_last_stats_update	Marca de tiempo para calcular estadísticas por segundo.
_processed_count	Contador de paquetes procesados.
_dropped_count	Contador de paquetes descartados.

Tabla 10. Clase Router atributos def__init__.

```
43    def __init__(self, name: str):
44        self.name = name
45        self.interfaces: Dict[str, Interface] = {}
46        self.routing_table = {}
47        self.qos_policy = QoSPolicy.FIFO
48        self.queues = {
49            TrafficType.VOICE: deque(),
50            TrafficType.VIDEO: deque(),
51            TrafficType.DATA: deque()
52        }
53        self.traffic_priorities = {
54            TrafficType.VOICE: 1,  # Mayor prioridad
55            TrafficType.VIDEO: 2,
56            TrafficType.DATA: 3    # Menor prioridad
57        }
58        self.stats = {
59            'processed_packets': 0,
60            'dropped_packets': 0,
61            'total_latency': 0.0,
62            'last_update_time': time.time(),
63            'processed_last_second': 0,
64            'dropped_last_second': 0,
65            'max_latency': 0.0,
66            'bandwidth_usage': 0.0,
67            'jitter': 0.0,
68            'queue_length': 0,
69            'packet_loss_rate': 0.0,
70            'last_latency': 0.0
71        }
72        self._last_stats_update = time.time()
73        self._processed_count = 0
74        self._dropped_count = 0
```

Imagen 8. Clase Router def __init__.

- **def add_interface**

  Se agrega una interfaz al router con un nombre y una capacidad máxima de la interfaz.

```
76    def add_interface(self, name: str, capacity: int):
77        self.interfaces[name] = Interface(capacity)
```

Imagen 9. Clase Router def add_interface.

- **def process_packet**

  Este método realizará la simulación del procesamiento de un paquete. Lo representaremos a través de un diagrama de flujo *(Imagen 10)*.

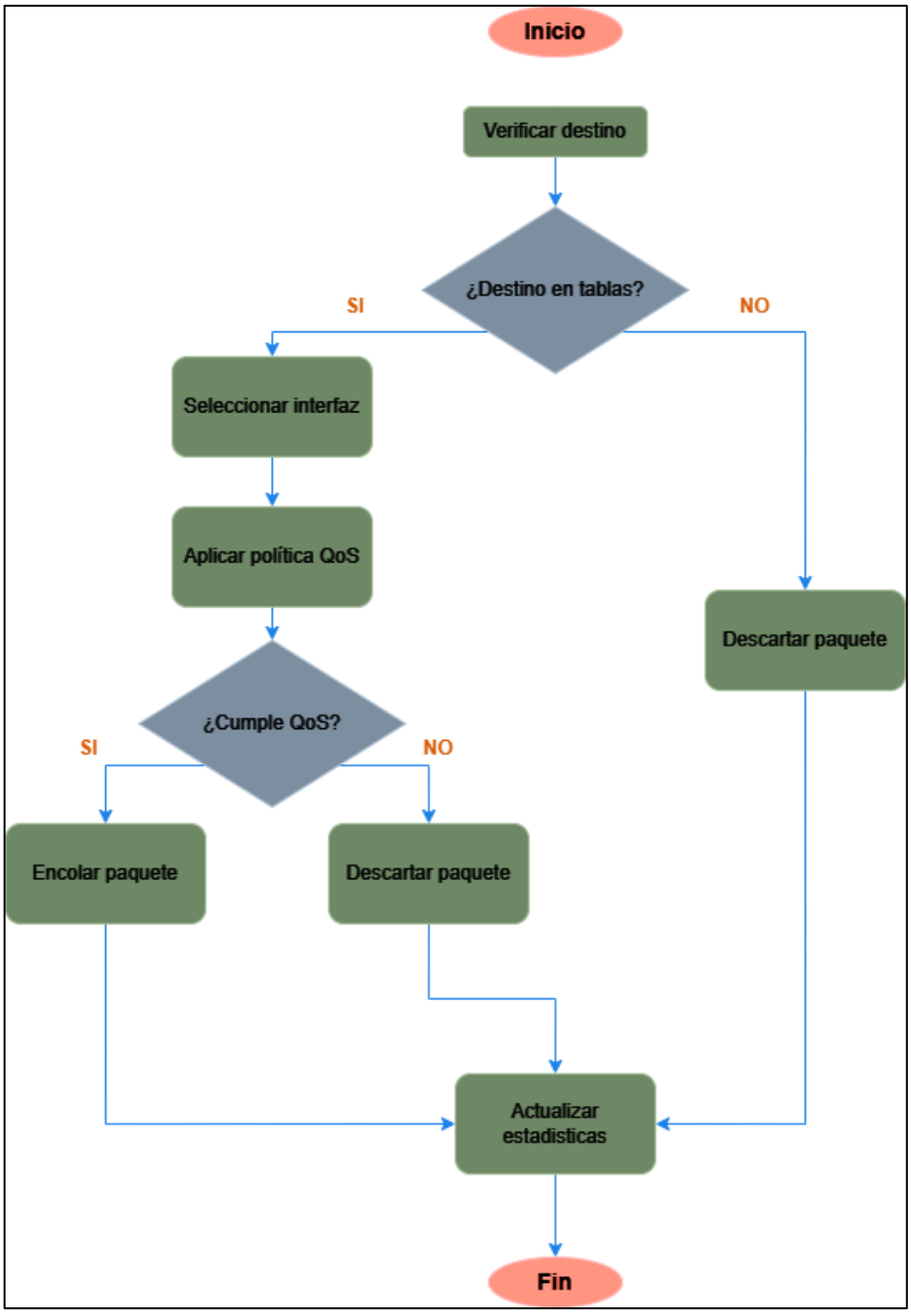


Imagen 10. Diagrama de flujo def process_packet.

```python
def process_packet(self, packet: Packet) -> bool:
    # Simular procesamiento de paquete
    if packet.destination in self.routing_table:
        interface_name = self.routing_table[packet.destination]
        interface = self.interfaces[interface_name]

        # Simular decisión de dropping basada en QoS
        dropped = False
        queue_size = len(interface.output_queue)

        if self.qos_policy == QoSPolicy.RED:
            # Random Early Detection
            if queue_size > 50:  # Umbral
                drop_probability = (queue_size - 50) / 50
                dropped = random.random() < drop_probability

        if not dropped and queue_size >= 100:  # Tamaño máximo de cola
            dropped = True

        # Actualizar estadísticas
        self.update_stats(packet, dropped, queue_size)

        if not dropped:
            interface.output_queue.append(packet)
            return True

    return False
```

Imagen 11. Clase Router def process_packet.

- **def update_stats**

  Actualiza las estadísticas del router como latencia, jitter, paquetes procesados, paquetes descartados, etc *(Tabla 11).*

Estadística	Descripción
total_latency	Tiempo acumulado de procesamiento de paquetes.
max_latency	Máxima latencia registrada.
jitter	Variación de latencia entre paquetes consecutivos.
procesed_last_second	Paquetes procesados en el último segundo.
dropped_last_second	Paquetes descartados en el último segundo.
packet_loss_rate	Porcentaje de paquetes descartados respecto al total recibido.
bandwidth_usage	Uso de ancho de banda basado en el procesamiento de paquetes.

Tabla 11. Clase Router def update_stats.

```python
def update_stats(self, packet=None, dropped=False, queue_size=0):
    current_time = time.time()
    time_diff = current_time - self._last_stats_update

    if packet:
        latency = current_time - packet.creation_time
        self.stats['total_latency'] += latency
        self.stats['max_latency'] = max(self.stats['max_latency'], latency)

        # Calcular jitter
        if self.stats['last_latency'] > 0:
            self.stats['jitter'] = abs(latency - self.stats['last_latency'])
        self.stats['last_latency'] = latency

    if dropped:
        self._dropped_count += 1
    else:
        self._processed_count += 1

    # Actualizar estadísticas por segundo si ha pasado al menos 1 segundo
    if time_diff >= 1.0:
        self.stats['processed_last_second'] = self._processed_count / time_diff
        self.stats['dropped_last_second'] = self._dropped_count / time_diff

        # Reiniciar contadores
        self._processed_count = 0
        self._dropped_count = 0
        self._last_stats_update = current_time

    # Actualizar estadísticas generales
    self.stats['processed_packets'] += 0 if dropped else 1
    self.stats['dropped_packets'] += 1 if dropped else 0
    total_packets = self.stats['processed_packets'] + self.stats['dropped_packets']

    if total_packets > 0:
        self.stats['packet_loss_rate'] = (self.stats['dropped_packets'] / total_packets) * 100

    # Actualizar uso de ancho de banda y longitud de cola
    self.stats['queue_length'] = queue_size
    self.stats['bandwidth_usage'] = (self._processed_count / max(time_diff, 1.0)) * 100
```

Imagen 12. Clase Router def update_stats.

- **def get_statistics**

  Retorna un resumen de las estadísticas actuales, permitiéndonos monitorear el desempeño del enrutador en tiempo real del número de paquetes procesados y descartados, latencia promedio y máxima, jitter, uso del ancho de banda y la tasa de pérdida de paquetes *(Tabla 12)*.

Estadística	Descripción
processed	Total de paquetes procesados.
dropped	Total de paquetes descartados.
latency	Latencia promedio.
max_latency	Máxima latencia registrada.
bandwidth_usage	Porcentaje de uso del ancho de banda.
jitter	Variación de latencia.
queue_length	Longitud actual de la cola.
packet_loss_rate	Porcentaje de paquetes descartados.

Tabla 12. Clase Router def get_statistics.

```python
    def get_statistics(self):
        """Retorna las estadísticas del router."""
        try:
            avg_latency = (self.stats['total_latency'] /
                           max(self.stats['processed_packets'], 1))
        except ZeroDivisionError:
            avg_latency = 0

        return {
            'processed': self.stats['processed_packets'],
            'dropped': self.stats['dropped_packets'],
            'latency': avg_latency,
            'processed_per_sec': self.stats['processed_last_second'],
            'dropped_per_sec': self.stats['dropped_last_second'],
            'max_latency': self.stats['max_latency'],
            'bandwidth_usage': self.stats['bandwidth_usage'],
            'jitter': self.stats['jitter'],
            'queue_length': self.stats['queue_length'],
            'packet_loss_rate': self.stats['packet_loss_rate']
        }
```

Imagen 13. Clase Router def get_statistics

class NetworkSimulator

- **def __init__**

  Se inicializa una nueva instancia de la clase NetworkSimulator, creando un contenedor donde se almacenará y organizará los router permitiéndonos gestionar múltiples routers de manera correcta.

```python
    def __init__(self):
        self.routers: Dict[str, Router] = {}
```

Imagen 14. Clase NetworkSimulator def __init__.

- **def add_router**

  Crea un router, dandole un nombre y lo añade automáticamente al contenedor central, de esta forma permite añadir dinámicamente nuevos enrutadores al simulador de red.

```python
    def add_router(self, name: str) -> Router:
        router = Router(name)
        self.routers[name] = router
        return router
```

Imagen 15. Clase NetworkSimulator def add_router.

- **def get_router**

  Busca el router por su nombre en el contenedor. Si existe, devuelve el objeto; de lo contrario, retorna None.

```
177
178     def get_router(self, name: str) -> Router:
179         return self.routers.get(name)
```

Imagen 16. Clase NetworkSimulator def get_router.

3.2.1.2. CAPA DE PRESENTACIÓN

Esta GUI actúa como la capa de presentación, facilitando la interacción entre el usuario y la capa de simulación, es decir la parte del sistema que el usuario ve y usa para interactuar con la herramienta permitiéndole seleccionar un límite de routers y aplicar las configuraciones QoS, además ver estadísticas en tiempo real de la cantidad de datos circulando por la red. La interfaz está organizada en pestañas para facilitar el acceso a diferentes funcionalidades *(Tabla 13)*.

Pestaña	Descripción	Funcionalidad
Principal	Monitoreo en tiempo real de la simulación.	• Estadísticas en tiempo real • Visualización de graficas
Configuración	Ajustes a los routers.	• Asignación de políticas QoS a routers
Análisis Detallado	Análisis profundo de estadísticas.	• Estadísticas avanzadas • Generación y visualización de reportes
Resultados	Resultados gráficos y textuales obtenidos de la simulación.	• Analizar el rendimiento de la red mediante métricas como latencia, jitter, pérdida de paquetes y uso de ancho de banda.

Tabla 13. GUI. Pestañas.

3.2.1.2.1. PESTAÑA PRINCIPAL

Es el centro principal del simulador, aquí tenemos una vista inmediata y completa del estado de la red. Esta pestaña está diseñada para ofrecer visibilidad en tiempo real de todos los parámetros del sistema. Tenemos un panel de control con 3 botones (Iniciar Simulación, Detener Simulación y Reiniciar Simulación) para facilitar el acceso rápido y gestionar la simulación, además de un indicador de estado (En ejecución y Detenido), también tenemos estadísticas en tiempo real de tres routers donde cada router tiene sus métricas de forma detallada incluyendo paquetes procesados, descartados, latencia, ancho de banda y jitter, En la parte inferior presentamos dos graficas de monitoreo en tiempo real de la Variación de Latencia y Longitud de Cola *(Imagen 17)*.

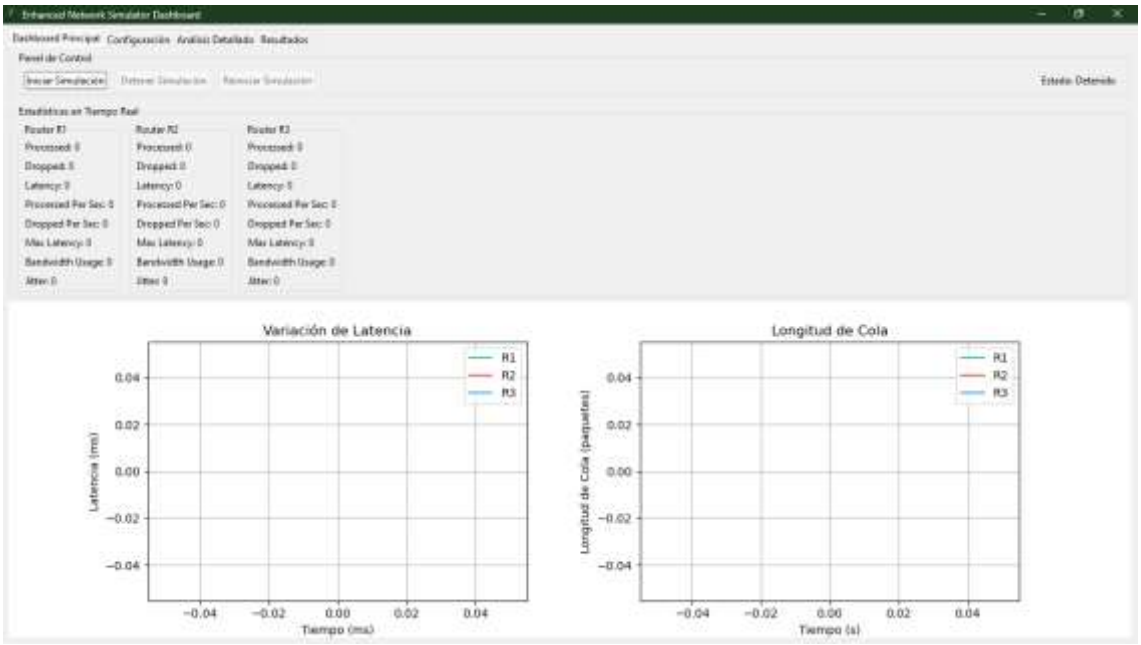


Imagen 17. GUI_Pestaña Principal.

3.2.1.2.2. PESTAÑA DE CONFIGURACIÓN

Esta pestaña muestra una interfaz para la configuración de QoS (Quality of Service), en la parte superior podemos incluir y configurar tres routers (R1, R2 y R3) mediante casillas de verificación, cada uno con un menú desplegable que ofrece diferentes opciones de gestión de cola (No QoS, FIFO, WFQ y RED), también podemos asignarle la cantidad de paquetes que serán procesados por cada router *(Imagen 18)*.

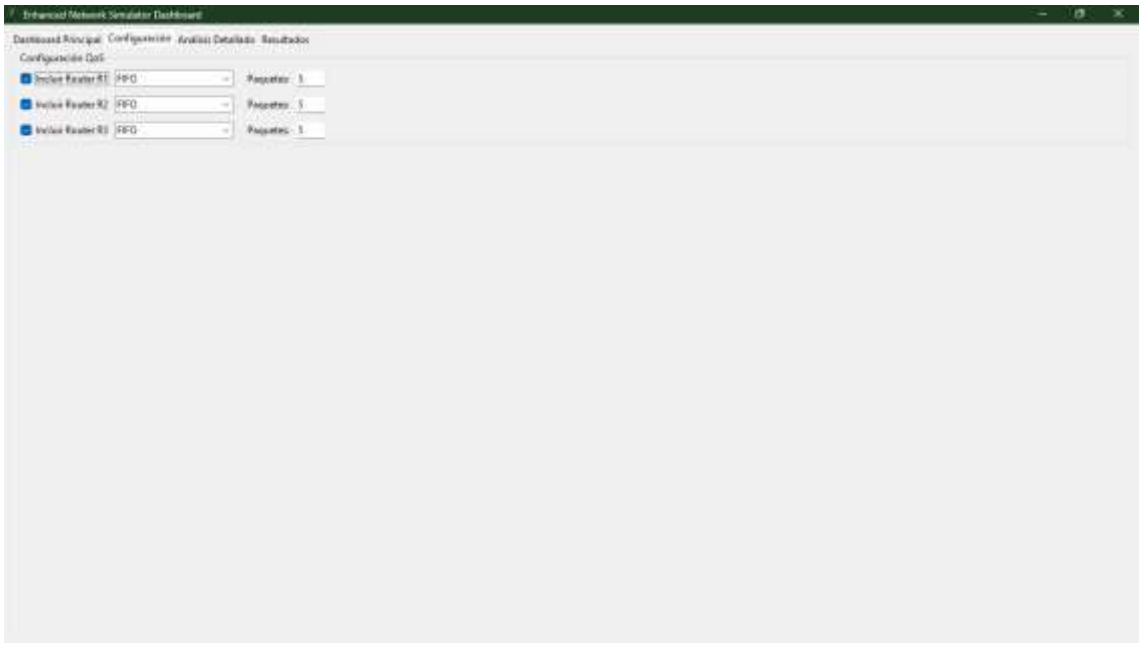


Imagen 18. GUI_Pestaña de Configuración.

3.2.1.2.3. PESTAÑA DE ANÁLISIS DETALLADO

En esta pestaña podemos ver diferentes métricas y visualizaciones de rendimiento. Tenemos 4 graficas que nos permiten analizar elementos claves:

1. **Gráfico de Vaciacion de Jitter:** este gráfico muestra la variación de la fluctuación a lo largo del tiempo, es decir podemos ver la variabilidad en el retraso de los paquetes de datos, que puede afectar a las aplicaciones en tiempo real.
2. **Gráfico de Uso de Ancho de Banda:** este gráfico muestra el uso del ancho de banda a lo largo de la simulación, nos muestra el porcentaje del ancho de banda que se está utilizando.
3. **Gráfico de Tasa de Pérdida de Paquetes:** este gráfico representa la tasa de pérdida de paquetes, es decir el porcentaje de paquetes de datos que no llegan a su destino siendo un impacto significativo en la calidad de los servicios de red.
4. **Gráfico de Índice de Rendimiento de Red:** este gráfico muestra la eficacia con la que la red utiliza el ancho de banda disponible, los valores más altos indican un mejor rendimiento de la red.

Estas visualizaciones brindan una descripción general completa de las características de rendimiento de la red, lo que permite al usuario identificar posibles problemas, cuellos de botella o áreas de optimización *(Imagen 19)*.

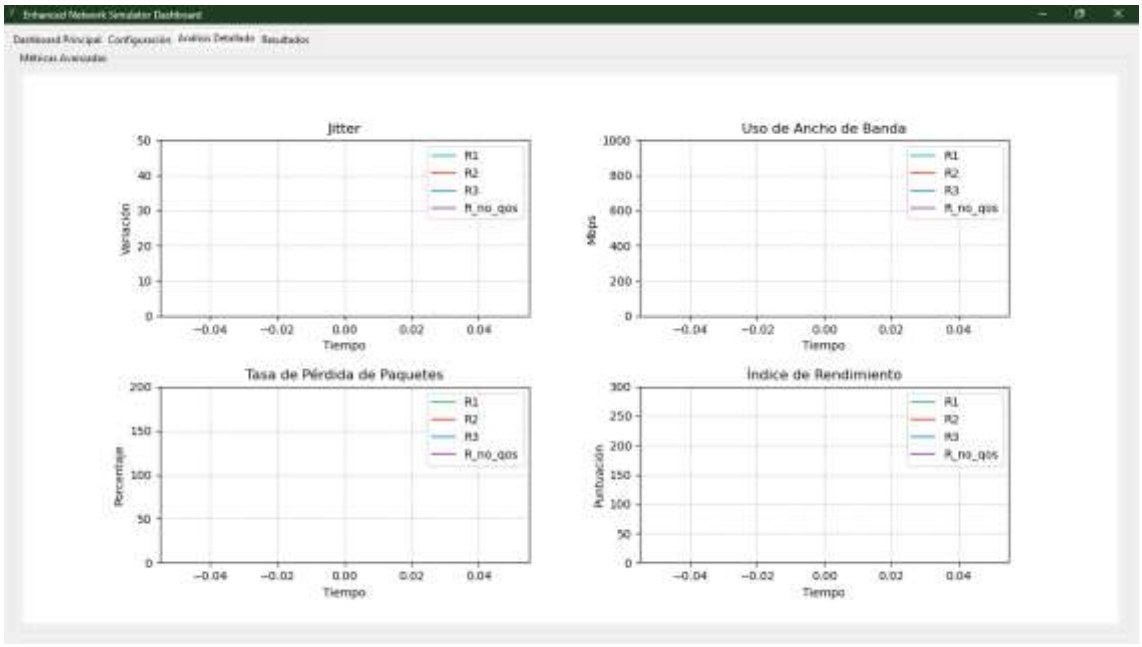


Imagen 19. GUI_Pestaña de Análisis Detallado.

3.2.1.2.4. PESTAÑA DE RESULTADOS

En la parte superior tenemos "Detalles de la Simulación" donde nos muestra información de métricas de la simulación de manera textual. En la parte inferior tenemos las "Graficas de Resultados" donde visualizaremos el rendimiento de la red, para analizar el comportamiento de la red.

Imagen 20. GUI_Pestaña de Resultados

FLUJO DE TRABAJO DE SIMULACIÓN DE RED

El usuario definirá la configuración de los routers asignándole políticas de Calidad de Servicio (QoS), a continuación, se generará tráfico con paquetes con atributos como tipo de tráfico, prioridad y destino. Durante el procesamiento de paquetes los routers aplican las políticas QoS según se le haya asignado para manejar el tráfico y organizar las colas según su tipo. Durante la simulación, se lleva a cabo una recopilación de datos, actualizando dinámicamente las estadísticas de rendimiento. Finalmente, en la etapa de visualización y análisis, el sistema presenta los resultados al usuario mediante gráficas donde observaremos métricas clave en tiempo real para facilitar la evaluación del desempeño de la red simulada por cada router *(Imagen 21)*.

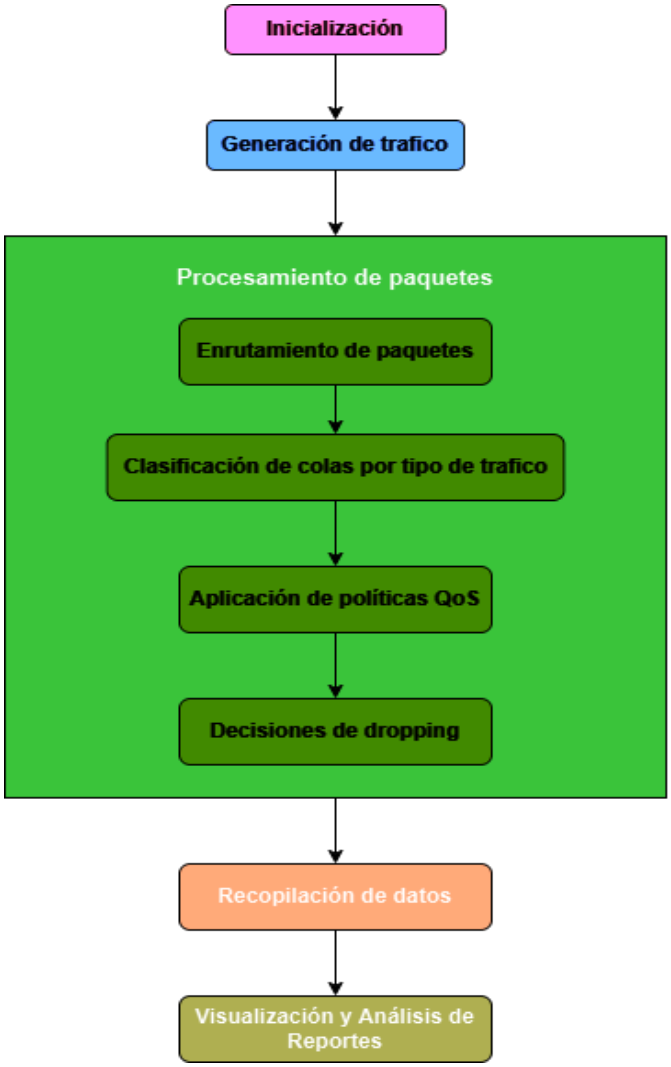


Imagen 21. Diagrama de flujo_ Simulación de red.

3.3. FASE DE PRUEBAS

<table>
<tr><th colspan="2">Prueba 1: Comparación de Rendimiento entre Router sin QoS y Router con política FIFO</th></tr>
<tr><td>Objetivo:</td><td>Comparar el rendimiento entre un router sin QoS y un router con política FIFO bajo tráfico moderado.</td></tr>
<tr><td>Descripción:</td><td>Analizar métricas como latencia, jitter, pérdida de paquetes, ancho de banda y longitud de cola en un entorno con tráfico moderado.</td></tr>
<tr><th colspan="2">Escenario 1: Tráfico Moderado</th></tr>
<tr><td>Datos de entrada:
Número de paquetes: 100
Número de routers: 2
Tipos de tráfico: Mixto (datos, voz, video).</td><td>Resultados observados:
La variación de jitter es estable cerca de 0 en ambos routers, sin diferencias significativas entre R1(No QoS) y R2(FIFO).
El uso de ancho de banda fluctúa entre 0 y 400 Mbps, con R2(FIFO) mostrando picos más altos, mientras que R1(No QoS) mantiene un uso más constante.
El índice de rendimiento es de 61.78% para R1(No QoS) y de 70.92% para R2(FIFO).</td></tr>
</table>

Tabla 14. Prueba1_Comparación de Rendimiento entre Router sin QoS y Router con política FIFO.

<table>
<tr><th colspan="2">Prueba 2: Comparación de Rendimiento entre Router sin QoS y Router con política RED</th></tr>
<tr><td>Objetivo:</td><td>Evaluar cómo la política RED mejora el manejo del tráfico en comparación con un router sin QoS bajo condiciones de tráfico moderado.</td></tr>
<tr><td>Descripción:</td><td>Se analizarán las métricas clave para determinar cómo RED reduce la congestión en comparación con un router sin QoS.</td></tr>
<tr><th colspan="2">Escenario 1: Tráfico Moderado</th></tr>
<tr><td>Datos de entrada:
Número de paquetes: 100
Número de routers: 2
Tipos de tráfico: Mixto (datos, voz, video).</td><td>Resultados observados:
La variación de jitter se mantiene relativamente baja para los routers R1(No QoS) y R2(RED).
En Router 1 (No QoS), el uso de ancho de banda es irregular, mientras que en Router 2 (RED) es más estable.
En Router 1 (No QoS), la pérdida de paquetes es mayor, mientras que en Router 2 (RED) es menor.
El índice de rendimiento es de 61.57% para R1(No QoS) y de 62.22% para R2(RED).</td></tr>
</table>

Tabla 15. Prueba2_Comparación de Rendimiento entre Router sin QoS y Router con política RED.

<table>
<tr><th colspan="2">Prueba 3: Comparación de Rendimiento entre Router sin QoS y Router con política WFQ</th></tr>
<tr><td>Objetivo:</td><td>Comparar el rendimiento de un router con política WFQ frente a uno sin QoS, analizando la priorización de tráfico bajo condiciones moderadas.</td></tr>
<tr><td>Descripción:</td><td>Analizar cómo WFQ prioriza eficientemente el tráfico mixto y reduce la latencia en comparación con un router sin QoS.</td></tr>
<tr><th colspan="2">Escenario 1: Tráfico Moderado</th></tr>
<tr><td>Datos de entrada:
Número de paquetes: 100
Número de routers: 2
Tipos de tráfico: Mixto (datos, voz, video).</td><td>Resultados observados:
La variación de jitter en ambos enrutadores es mínima y se mantiene cerca de 0 durante toda la simulación, lo que indica un tiempo de entrega de paquetes estable en ambas configuraciones.
El uso de ancho de banda fluctúa hasta aproximadamente 400 Mbps, con R2 (WFQ) mostrando picos ligeramente más altos en su utilización.
Las tasas de pérdida parecen oscilar entre el 0 y el 50%.
El índice de rendimiento es de 65.73% para R1(No QoS) y de 73.57% para R2(WFQ).</td></tr>
</table>

Tabla 16. Prueba3_Comparación de Rendimiento entre Router sin QoS y Router con política WFQ.

<table>
<tr><th colspan="2">Prueba 4: Evaluación de políticas de Calidad de servicio (QoS) en routers</th></tr>
<tr><td>Objetivo:</td><td>Analizar el impacto de las políticas de calidad de servicio (QoS) FIFO, WFQ y RED en el rendimiento de una red con tráfico mixto (datos, voz y video).</td></tr>
<tr><td>Descripción:</td><td>Se implementarán tres routers, cada uno con una política QoS distinta (FIFO, WFQ, RED). Los datos de entrada y salida serán monitoreados para medir latencia, jitter, ancho de banda, pérdida de paquetes y variación en la longitud de cola.</td></tr>
<tr><th colspan="2">Escenario 1: Tráfico Bajo</th></tr>
<tr><td>Datos de entrada:
Número de paquetes: 50
Número de routers: 3
Tipos de tráfico: Mixto (datos, voz, video).</td><td>Resultados observados:
El Jitter se mantiene estable y cercano a 0 para los tres routers.
El ancho de banda muestra un uso moderado con algunas fluctuaciones.
La pérdida de paquetes presenta una tendencia creciente
Rendimiento del Router R1(FIFO) 68.77%
Rendimiento del Router R2(WFQ) 80.22% (Mejor rendimiento)
Rendimiento del Router R3(RED) 67.55%</td></tr>
</table>

Escenario 2: Tráfico Moderado	
Datos de entrada: • Número de paquetes: 100 • Número de routers: 3 • Tipos de tráfico: Mixto (datos, voz, video).	**Resultados observados:** • El Jitter se mantiene estable y cercano a 0 para los tres routers. • El ancho de banda fluctúa entre los 0-600 Mbps. • Rendimiento del Router R1(FIFO) 59.13% • Rendimiento del Router R2(WFQ) 62.91% (Mejor rendimiento) • Rendimiento del Router R3(RED) 60.72%
Escenario 3: Tráfico Alto	
Datos de entrada: • Número de paquetes: 200 • Número de routers: 3 • Tipos de tráfico: Mixto (datos, voz, video).	**Resultados observados:** • Jitter estable y muy bajo durante toda la simulación para todos los routers. • El ancho de banda fluctúa entre los 0-400Mbps. • La tasa de perdida de paquetes oscila entre 0-100% • Rendimiento del Router R1(FIFO) 68.57% • Rendimiento del Router R2(WFQ) 69.06% (Mejor rendimiento) • Rendimiento del Router R3(RED) 54.76% (Menor eficiencia)

Tabla 17. Evaluación de políticas QoS.

3.4. FASE DE ANÁLISIS Y VALIDACIÓN

3.4.1. RESULTADOS

Prueba 1: Comparación de Rendimiento entre Router sin QoS y Router con política FIFO

En el router R1 (sin QoS) y R2 (con política FIFO) observamos mejoras significativas al implementar QoS, aunque ambos mantienen una latencia promedio de 0.00 ms, pero R2 presenta menor pérdida de paquetes, un uso más eficiente del ancho de banda y un mayor índice de rendimiento promedio *(Tabla 18)*.

Métrica	R1 (No QoS)	R2 (FIFO)
Latencia promedio	0.00 ms	0.00 ms
Latencia máxima	0.00 ms	0.13 ms
Jitter promedio	0.00 ms	0.00 ms
Pérdida de paquetes	37%	29%
Uso de ancho de banda	286.728 Mbps	375.973 Mbps
Índice de rendimiento	61.78%	70.92%

Tabla 18. Comparación de Rendimiento entre Router sin QoS y Router con política FIFO.

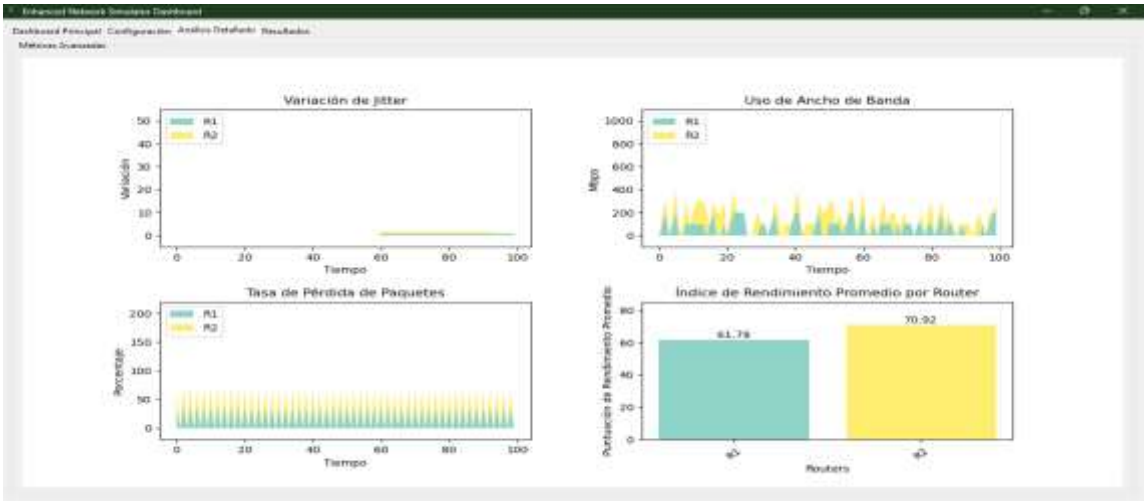


Imagen 22. Comparación de Rendimiento entre Router sin QoS y Router FIFO 1.

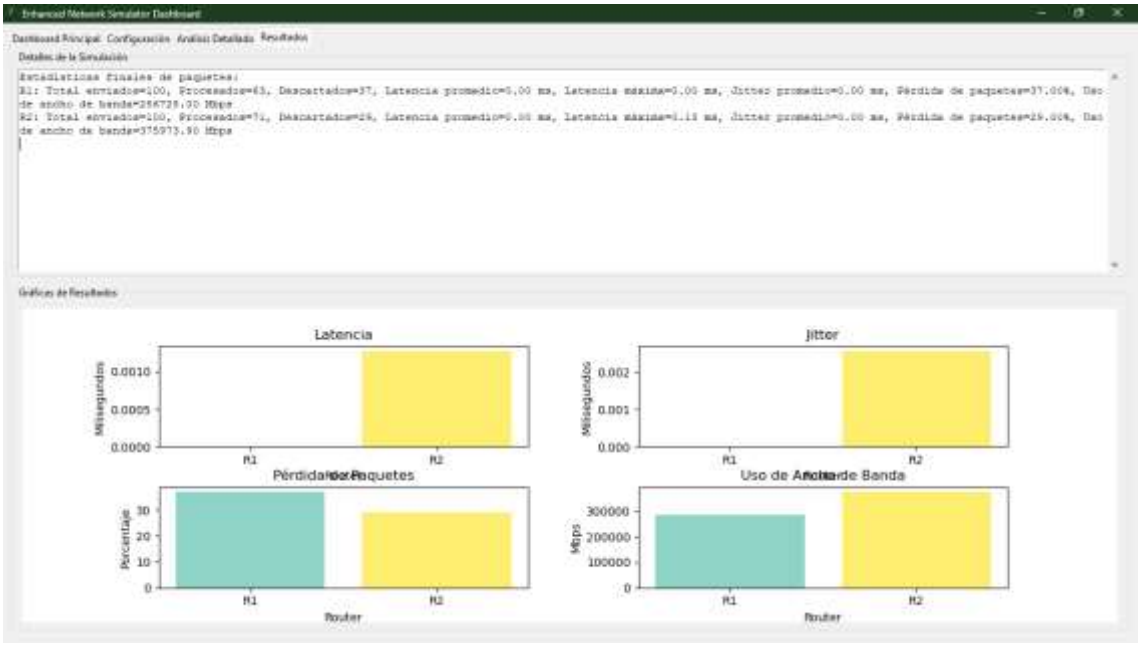


Imagen 23. Comparación de Rendimiento entre Router sin QoS y Router FIFO 2.

Prueba 2: Comparación de Rendimiento entre Router sin QoS y Router con política RED

En el router R2 (con política RED) observamos mejoras moderadas en algunos aspectos claves, como una menor pérdida de paquetes y un índice de rendimiento ligeramente superior. Sin embargo, estos beneficios vienen acompañados de una latencia promedio y máxima ligeramente mayores, así como un aumento mínimo en el jitter. Sin embargo, el router R1 (Sin QoS) destaca por un uso más eficiente del ancho de banda *(Tabla 19).*

Métrica	R1 (No QoS)	R2 (RED)
Latencia promedio	0.00 ms	0.01 ms
Latencia máxima	0.00 ms	0.51 ms
Jitter promedio	0.00 ms	0.01 ms
Pérdida de paquetes	35%	32%
Uso de ancho de banda	296.664 Mbps	284.249 Mbps
Índice de rendimiento	61.57%	62.22%

Tabla 19. Comparación de Rendimiento entre Router sin QoS y Router con política RED.

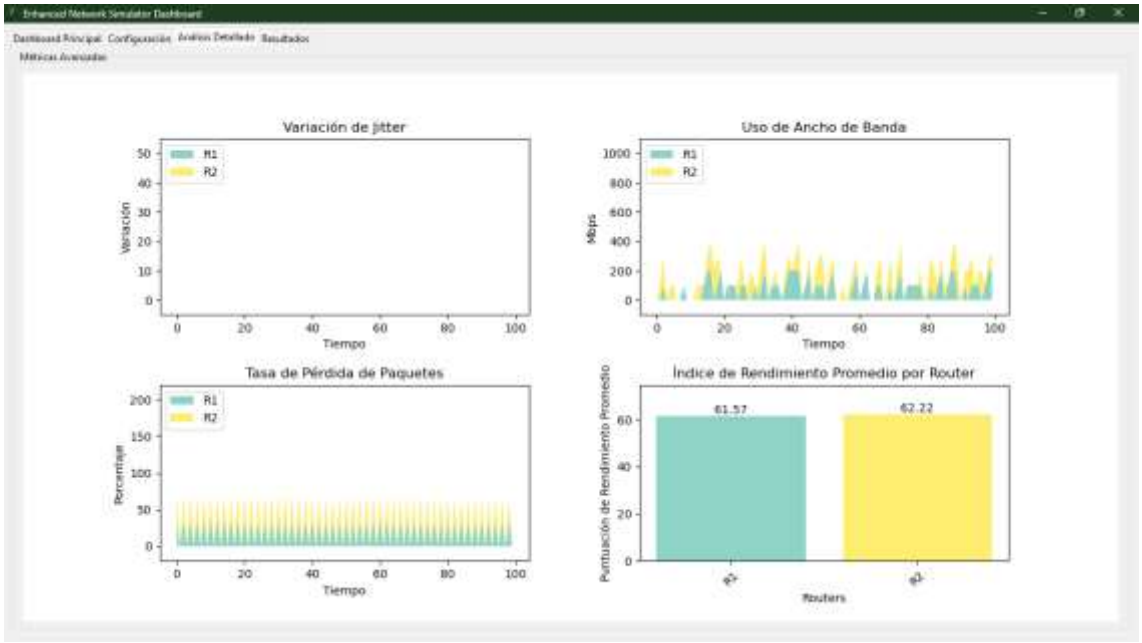


Imagen 24. Comparación de Rendimiento entre Router sin QoS y Router con política RED 1.

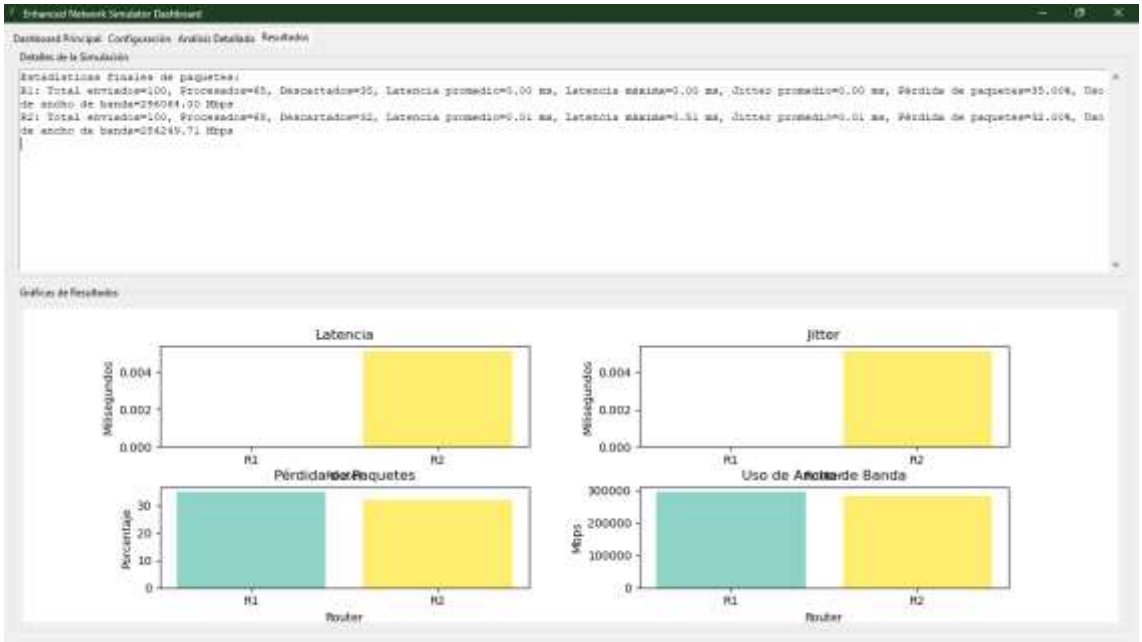


Imagen 25. Comparación de Rendimiento entre Router sin QoS y Router con política RED 2.

Prueba 3: Comparación de Rendimiento entre Router sin QoS y Router con política WFQ

La simulación entre el Router 1 (sin QoS) y el Router 2 (con política WFQ) demuestra mejoras notables en la gestión del tráfico de red. El Router 2 reduce significativamente la pérdida de paquetes, mejora el rendimiento y optimiza el uso del ancho de banda, todo ello sin comprometer la estabilidad, manteniendo latencia y jitter en niveles óptimos.

Métrica	R1 (No QoS)	R2 (WFQ)
Latencia promedio	0.00 ms	0.00 ms
Latencia máxima	0.00 ms	0.00 ms
Jitter promedio	0.00 ms	0.00 ms
Pérdida de paquetes	31%	23%
Uso de ancho de banda	294.264 Mbps	250.008 Mbps
Índice de rendimiento	65.73%	73.57%

Tabla 20. Comparación de Rendimiento entre Router sin QoS y Router con política WFQ.

Imagen 26. Comparación de Rendimiento entre Router sin QoS y Router con política WFQ 1.

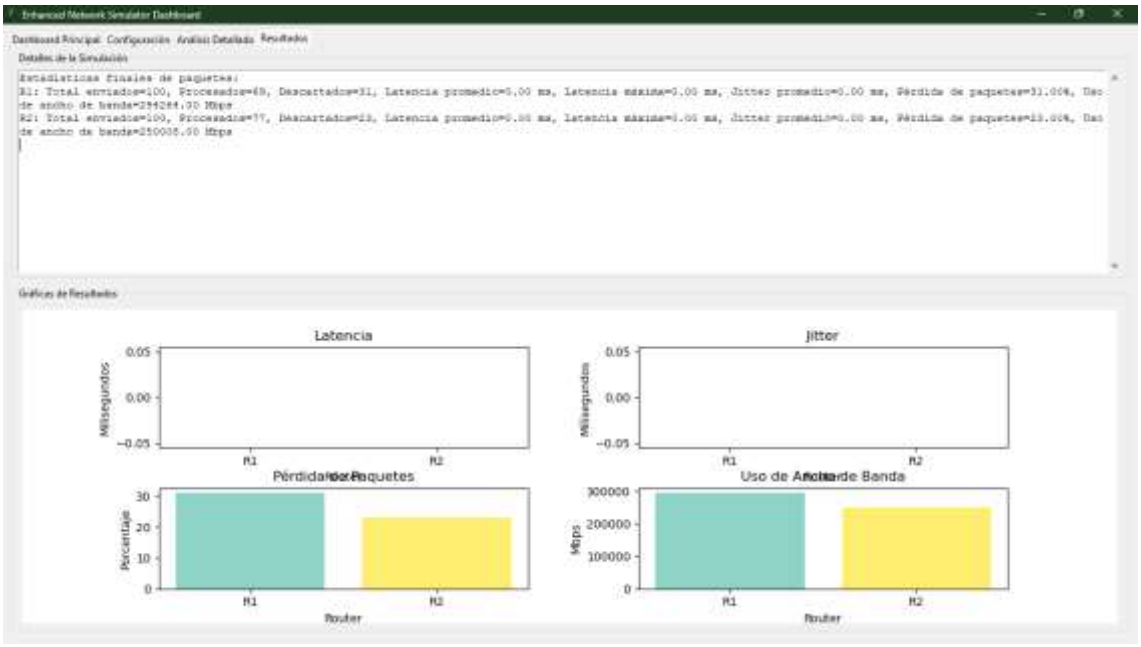


Imagen 27. Comparación de Rendimiento entre Router sin QoS y Router con política WFQ 2.

Prueba 4: Evaluación de políticas de Calidad de servicio (QoS) en routers

Tráfico Bajo: Las políticas de QoS muestran resultados variados. WFQ se destaca con el mejor rendimiento general, aunque usa un poco más de ancho de banda. RED es más eficiente al reducir la pérdida de paquetes, mientras que FIFO, aunque más simple, mantiene un desempeño aceptable para tareas básicas. Todos los routers logran una transmisión estable con latencia y jitter óptimos *(Tabla 21)*.

Métrica	R1 (FIFO)	R2 (WFQ)	R3 (RED)
Latencia promedio	0.00 ms	0.00 ms	0.00 ms
Latencia máxima	0.00 ms	0.00 ms	0.00 ms
Jitter promedio	0.00 ms	0.00 ms	0.00 ms
Pérdida de paquetes	42%	34%	26%
Uso de ancho de banda	128.936 Mbps	143.320 Mbps	140.416 Mbps
Índice de rendimiento	68.77%	80.22%	67.55%

Tabla 21. Evaluación de políticas de Calidad de servicio (QoS) en routers_Tráfico bajo.

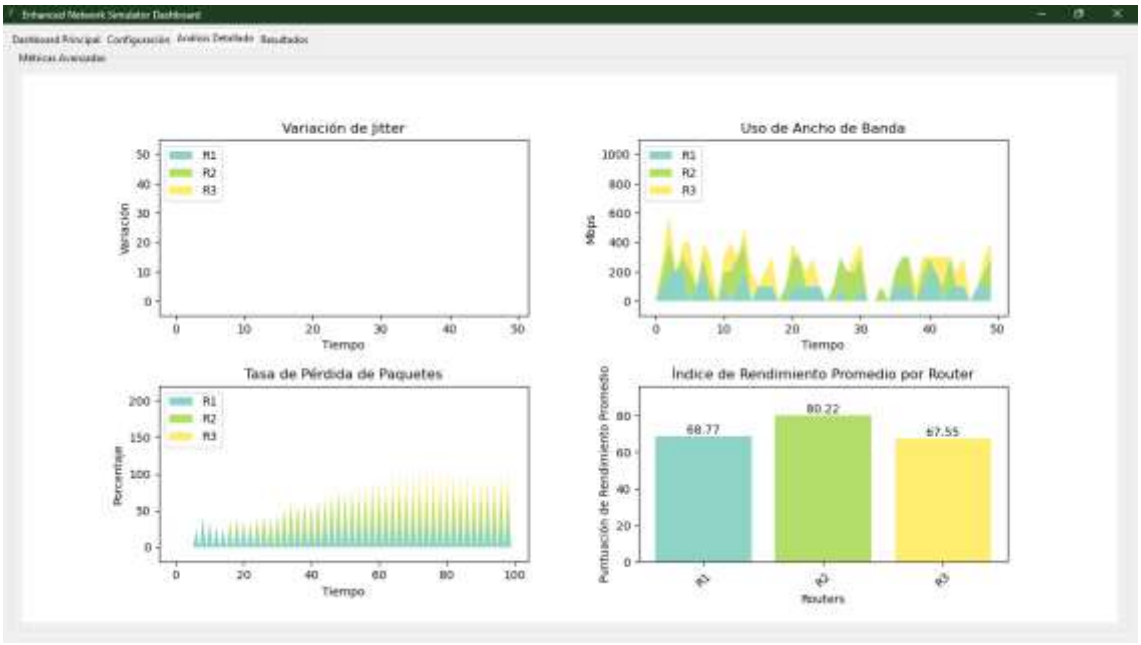


Imagen 28. Evaluación de políticas de Calidad de servicio (QoS) en routers E1.

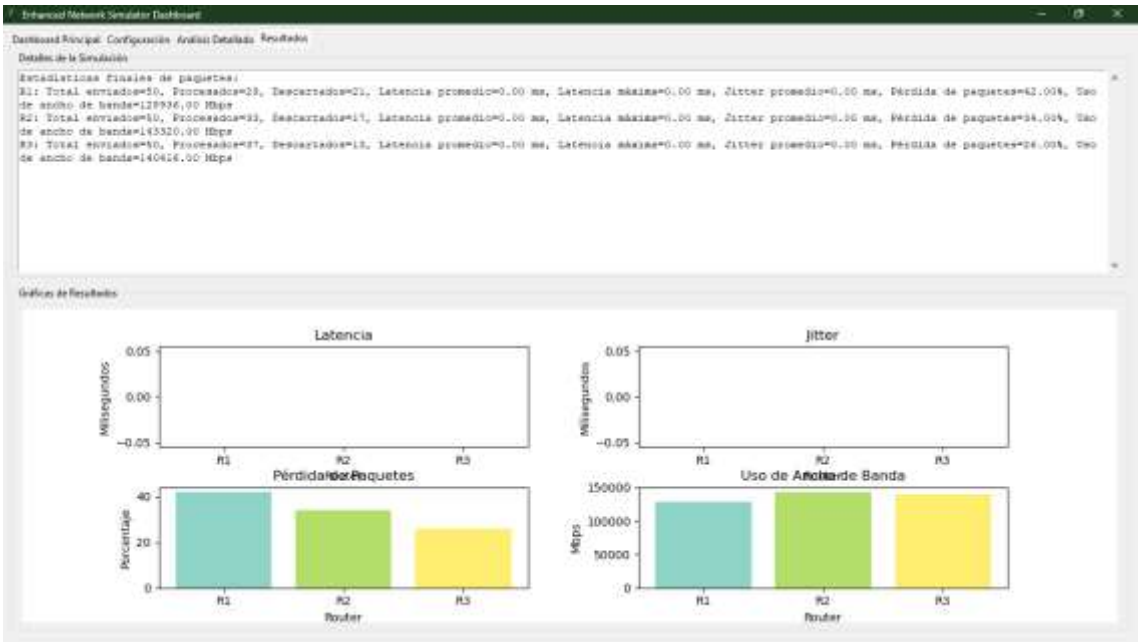


Imagen 29. Evaluación de políticas de Calidad de servicio (QoS) en routers E1_1.

Tráfico moderado: Podemos observar que no hay diferencias en la latencia ni en el jitter entre los tres routers. Sin embargo, el router R2 (WFQ) destaca con menor pérdida de paquetes, seguido por R3 (RED) y R1 (FIFO). Aunque R2 utiliza más ancho de banda, también alcanza el mejor índice de rendimiento, superando a los otros dos routers *(Tabla 22)*.

Métrica	R1 (FIFO)	R2 (WFQ)	R3 (RED)
Latencia promedio	0.00 ms	0.00 ms	0.00 ms
Latencia máxima	0.00 ms	0.00 ms	0.00 ms
Jitter promedio	0.00 ms	0.00 ms	0.00 ms
Pérdida de paquetes	39%	29%	32%
Uso de ancho de banda	200.816 Mbps	322.144 Mbps	206.496 Mbps
Índice de rendimiento	59.13%	62.21%	60.72%

Tabla 22. Evaluación de políticas de Calidad de servicio (QoS) en routers_Tráfico moderado.

Imagen 30. Evaluación de políticas de Calidad de servicio (QoS) en routers E2.

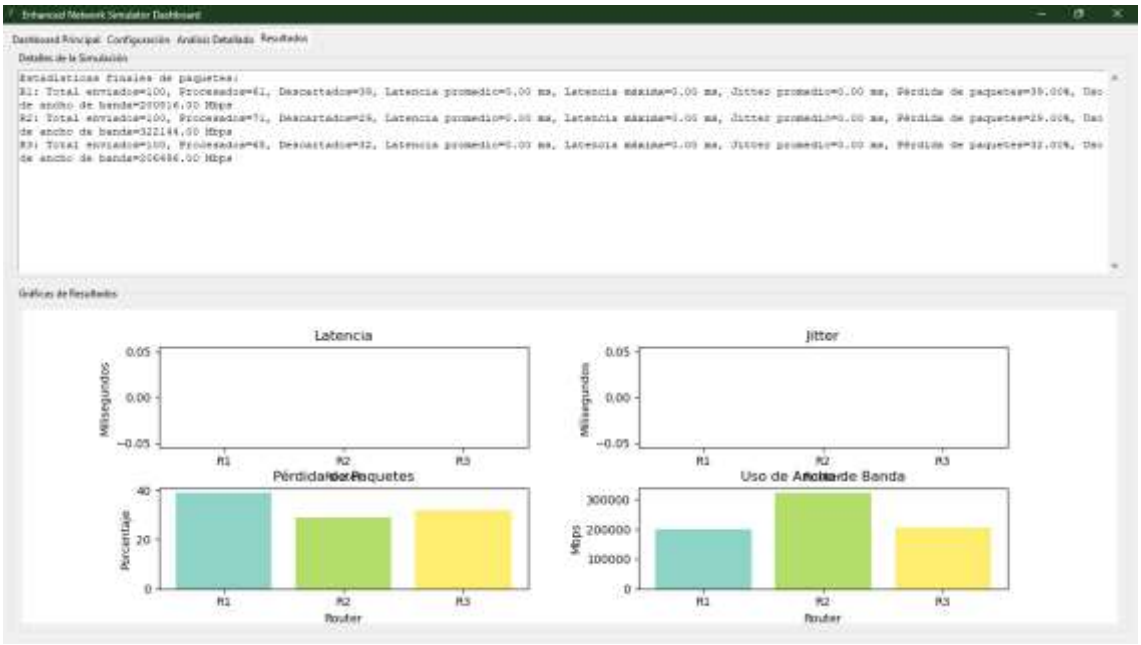


Imagen 31. Evaluación de políticas de Calidad de servicio (QoS) en routers E2_1.

Tráfico alto: Los tres routers muestran latencia y jitter similares, sin diferencias significativas en estos aspectos. En pérdida de paquetes, el router FIFO es el más eficiente, seguido por WFQ y luego RED. En uso de ancho de banda, FIFO consume más, mientras que RED es el más eficiente. En rendimiento, WFQ obtiene el mejor resultado, superando ligeramente a FIFO, con RED en último lugar *(Tabla 23)*.

Métrica	R1 (FIFO)	R2 (WFQ)	R3 (RED)
Latencia promedio	0.00 ms	0.00 ms	0.00 ms
Latencia máxima	0.00 ms	0.00 ms	0.00 ms
Jitter promedio	0.00 ms	0.00 ms	0.00 ms
Pérdida de paquetes	29.50%	33%	37.50%
Uso de ancho de banda	618.360 Mbps	533.272 Mbps	499.800 Mbps
Índice de rendimiento	68.57%	69.06%	54.76%

Tabla 23. Evaluación de políticas de Calidad de servicio (QoS) en routers_Tráfico alto.

Imagen 32. Evaluación de políticas de Calidad de servicio (QoS) en routers E3.

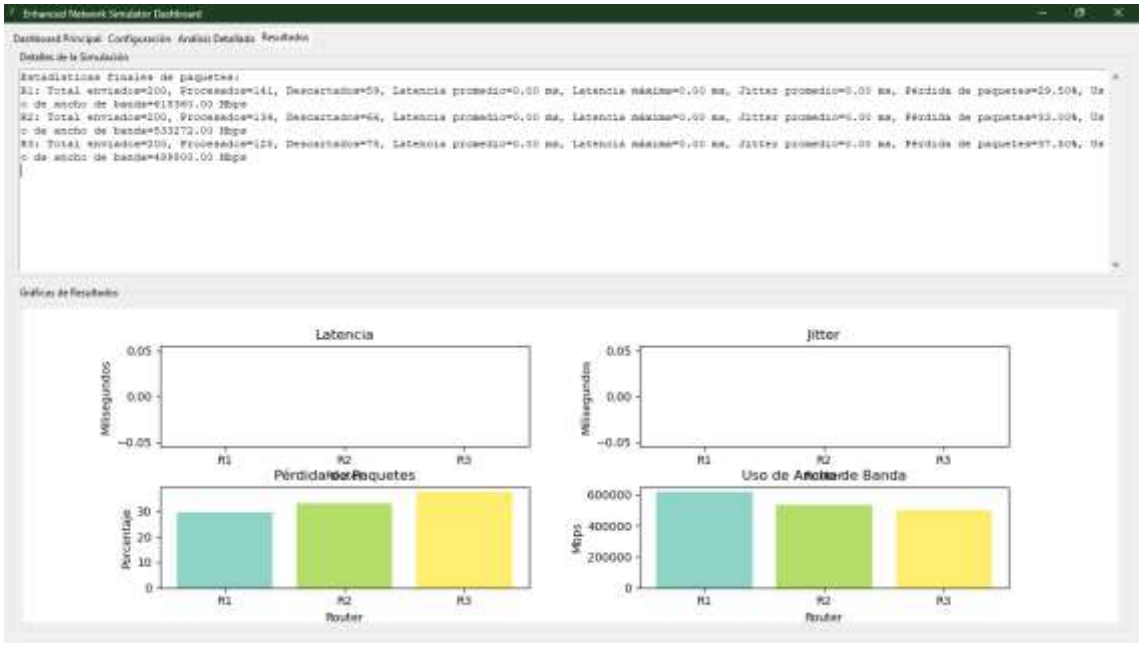


Imagen 33. Evaluación de políticas de Calidad de servicio (QoS) en routers E3_1.

3.5. FASE DE ENTREGA

3.5.1. GUIA DE USO DEL SIMULADOR

USO Y CONFIGURACIÓN DEL SIMULADOR

Al ejecutar “network_gui” en jupyter notebook nos aparecerá la ventana principal del simulador (*Imagen 34*), este es el interfaz principal del simulador.

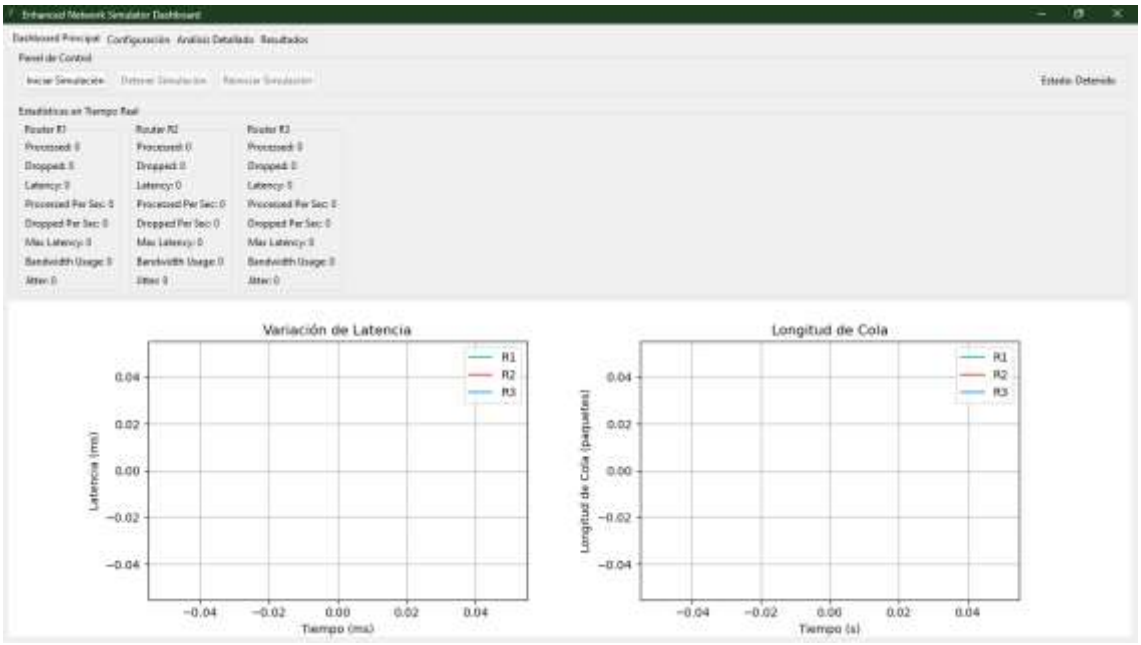


Imagen 34. GUIA - Ventana Principal

Para realizar las respectivas configuraciones de políticas QoS en los routers debemos dirigirnos a la pestaña de “Configuración” *(Imagen 35).* En esta ventana tenemos varios elementos configurables donde podremos habilitar y deshabilitar routers de acuerdo a las necesidades de cada usuario, además hay un menú desplegable que nos permite seleccionar diferentes política QoS para cada router, dentro de las opciones tenemos “No Qos”, esta opción deshabilita cualquier política de calidad de servicio para el router seleccionado, ideal para realizar simulaciones sin optimización de tráfico, también tenemos “FIFO” que procesa los paquetes en orden de llegada, “WFQ” que distribuye los recursos de la red según las prioridades establecidas y “RED” que detecta de forma temprana la congestión de paquetes para descartarlos de forma aleatoria y evitar la congestión en la red. Además, podemos asignar la cantidad de paquetes que serán procesados por cada router de manera individual.

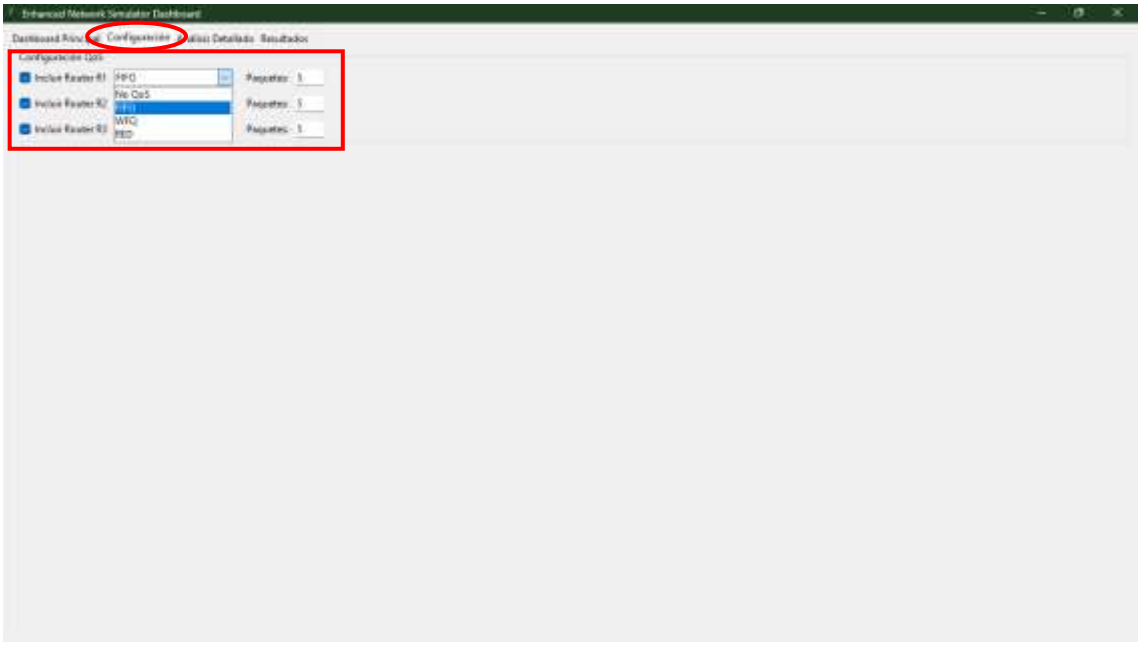


Imagen 35. GUIA - Ventana de Configuración.

Para darle inicio a la simulación debemos regresar a la ventana principal donde ya hechas las configuraciones respectivas y aplicadas podemos Iniciar la simulación. Luego de realizar el paso anterior podemos visualizar estadísticas en tiempo real de los routers de los paquetes procesados, paquetes descartados, latencia, paquetes procesados por segundo, paquetes descartados por segundo, latencia máxima, ancho de banda y jitter. En la parte inferior de la ventana principal podemos visualizar graficas en tiempo real de "Variación de latencia" y "Longitud de cola" *(Imagen 36)*.

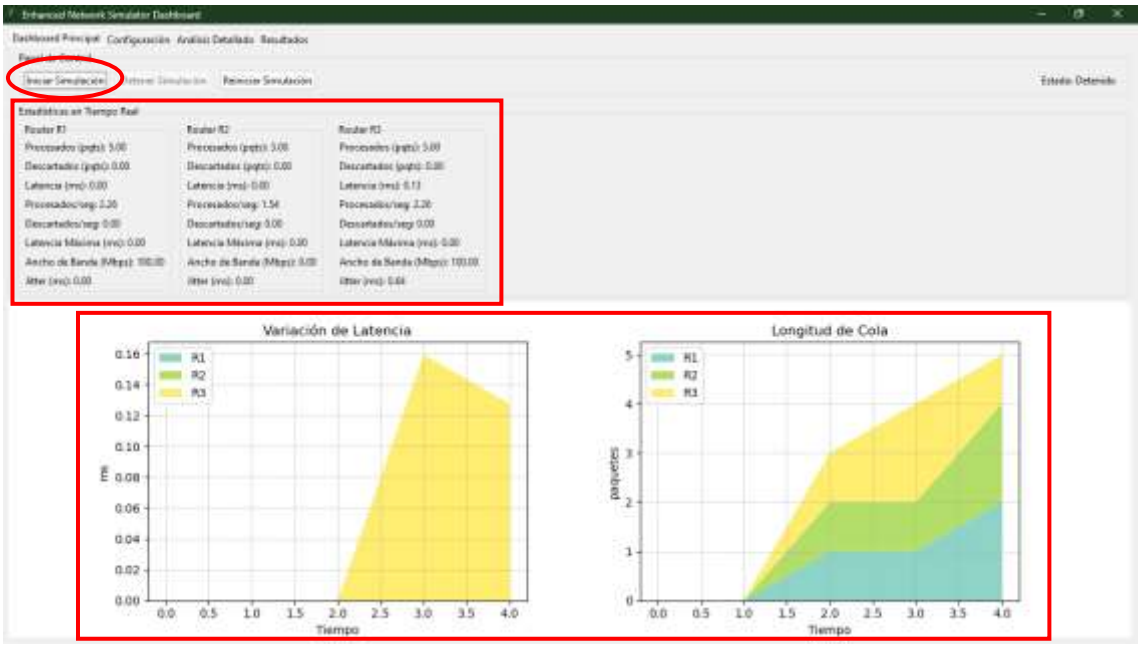


Imagen 36. GUIA - Ventana Principal Inicio de Simulacion.

Si queremos visualizar gráficas y poder realizar un análisis más detallado de la simulación de los routers de acuerdo a las políticas QoS configuradas podemos dirigirnos a la pestaña de “Análisis Detallado” donde tenemos una visualización detallada de las métricas clave para evaluar el rendimiento de la red simulada *(ver Imagen 37)*.

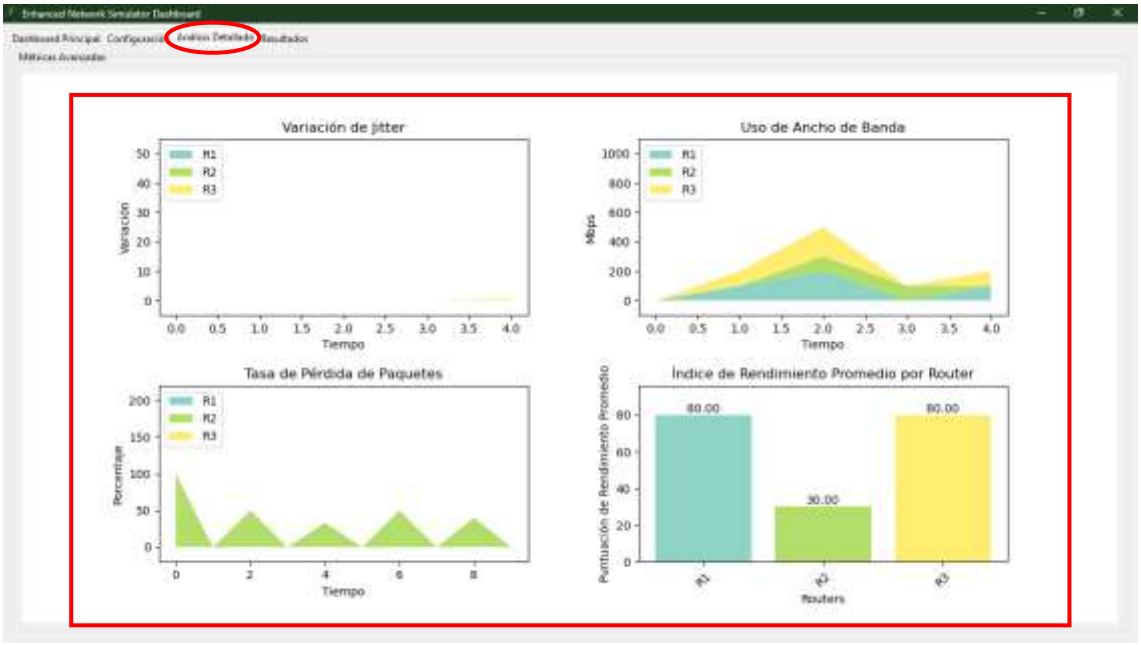


Imagen 37. GUIA - Ventana de Análisis Detallado.

En la pestaña de “Resultados” tenemos los “Detalles de simulación” donde se presenta un resumen textual con los datos obtenidos de la latencia promedio y máxima, el jitter, la pérdida de paquetes, indicada como un porcentaje y el uso del ancho de banda, que detalla cuántos Mbps fueron procesados por cada router. En la parte inferior teneos las “Graficas de Resultados” donde podemos ver la comparación visual del desempeño de los routers *(Imagen 38)*.

Imagen 38.GUIA – Ventana de Resultados.

CONCLUSIONES

- La implementación de las políticas QoS mostró mejoras significativas en el rendimiento de la red. Por primera vez, la política WFQ mostró la mejor tasa de rendimiento en todos los escenarios de tráfico, con un bajo de 80.22%, moderado de 62.91% y alto de 69.06%, superando a las políticas FIFO y RED.
- La política WFQ resultó ser la más efectiva en el manejo del tráfico mixto, con una reducción significativa de la pérdida de paquetes de 23% al no usar QoS al 31% y manteniendo la latencia y el jitter en 0ms, lo que resulta en la mejor calidad de servicio.
- La política RED presentó una mejora moderada en el rendimiento general, dado que tuvo un impacto más efectivo en un tráfico bajo y moderado la tasa (62.22% frente a 61.57% sin QoS), su rendimiento se degradó sustancialmente bajo condiciones de tráfico alto la tasa fue del 54.76%, lo cual fue la peor de todas las condiciones.
- La política FIFO apareció con el mejor desempeño en condiciones de tráfico alto fue de 68.57%, mejorando todas las otras metodologías y cerrando las brechas con WFQ, lo que sugiere que este es el método más eficiente en sistemas con recursos limitados.

RECOMENDACIONES

- Se recomienda la implementación de la política WFQ en entornos de red en los que se maneje tráfico mixto crítico, especialmente en escenarios en los que la pérdida de paquetes debe minimizarse y se necesita un alto rendimiento constante.
- Para redes con poco más que recursos o infraestructura básica, el uso de FIFO parece ser una buena opción aún, especialmente si el tráfico es en su mayoría de un solo tipo o si la red opera bajo condiciones de alta carga frecuentemente.
- En general, se aconseja no implementar políticas RED en escenarios de tráfico alto, ya que mostró bajo rendimiento. Por lo tanto, RED debiera implementarse solo en entornos de con poco a moderado tráfico en los que la prevención de la congestión tenga prioridad.
- Es prudente monitorear continuamente el ancho de banda, ya que, en las pruebas de rendimiento, hubo variabilidad significativa. Por lo tanto, es evidente que las políticas QoS deben ajustarse dinámicamente según la demanda.

BIBLIOGRAFÍA

[1] W. Stallings, COMUNICACIONES Y REDES DE COMPUTADORES. Séptima edición, Madrid: Pearson, 2004.

[2] E. Suntaxi, «Repositorio Institucional de la Universidad Politécnica Salesiana,» Febrero 2013. [En línea]. Available: https://dspace.ups.edu.ec/handle/123456789/18305.

[3] A. S. &. W. D. J. Tanenbaum, Computer Networks, Pearson, 2011.

[4] L. G. B. Almeida, «ANALISIS DE CALIDAD DE SERVICIO EN REDES,» Sangolqui.

[5] R. Moreano, «Metodología para evaluar la Calidad de Servicio,» Quito.

[6] D. F. V. Acuña, «RED INALÁMBRICA TIPO MALLA (WNM) ESTANDAR 802.11 DE,» 2014.

[7] O. J. Salcedo Parra, D. López y Á. P. Ríos, «Desempeño de la calidad del servicio (QoS) sobre IPv6,» Bogotá, 2011.

[8] B. Ram, «Advanced QoS for Multi-Service IP/MPLS Networks,» 2008.

[9] J. A. y. M. Sarmiento, «Implementación de QoS en la red LAN de la Universidad,» Loja, 2013.

[10] J. Becerra Guzmán y D. F. Arquez Beltrán, «MECANISMO DE MANEJO DE COLA EN REDES IP.,» CARTAGENA, 2008.

[11] C. Villalba, A. Moraleda y R. M., Lenguajes de programacion, España: UNED, 2021.

[12] Rootstack, «Los mejores entornos de desarrollo de Python para utilizar en el 2023,» Octubre 2022. [En línea]. Available: https://rootstack.com/es/blog/los-mejores-entornos-de-desarrollo-de-python-parautilizar-en-el-2023..

[13] S. Figueiras, «¿Conoces Jupyter Notebook?,» Lunes Septiembre 2021. [En línea]. Available: https://www.ceupe.mx/blog/conoces-jupyter-notebook.html.

[14] Bottega, «Qué es una librería en programación,» DevCamp, 2020.

[15] Python, «Pypi,» [En línea]. Available: https://pypi.org/. [Último acceso: 2024].

[16] W3Schools, «W3Schools,» [En línea]. Available: https://www.w3schools.com/. [Último acceso: 2024].

[17] Python, «Python,» 2024. [En línea]. Available: https://docs.python.org/3/library/index.html.

[18] Matplotlib, «Matplotlib 3.9.2 documentation,» [En línea]. Available: https://matplotlib.org/stable/tutorials/pyplot.html. [Último acceso: 2024].

[19] M. All, «Tutorial de Python Seaborn para principiantes: comience a visualizar datos,» 2023.

[20] DataScientest, «NumPy : La biblioteca de Python más utilizada en Data Science».

[21] Pandas, «pandas,» [En línea]. Available: https://pandas.pydata.org/. [Último acceso: 2024].

[22] L. R. Márquez, «Simulador en Python del procedimiento de predicción de la interferencia entre estaciones a frecuencias superiores a 0,1 GHz según la Rec. ITU-R P.452,» Sevilla, 2020.

[23] Z. Angélica, «Estudio de QoS sobre WLAN Utilizando el Estándar 802.11e a Transmisiones de Sistemas Multimediales en Tiempo Real,» Riobamba, 2010.

[24] Albentia, «Implementación de QoS en redes Wimax,» 2010.

[25] T. Braun, M. Diaz, J. Enríquez y T. Staub, «End-to-End Quality of Service Over Heterogeneous Networks,» 2008.

[26] S. Harpreet, «Implementing Cisco Networking Solutions,» 2017.

[27] A. Tanenbaum y D. Wetherall, «Redes de Computadoras,» 2016.

[28] R. Rojas, Guía para realizar investigaciones sociales., Mexico D.F.: Plaza y Valdés, 2013.

[29] A. R. Bueno, Noviembre 2015. [En línea]. Available: https://diposit.ub.edu/dspace/bitstream/2445/67618/1/EL_m%C3%A9todo_Experimental_conceptualizaci%C3%B3n.pdf.