uc3m | Universidad Carlos III de Madrid

Ingeniería Telemática

2024-2025

Trabajo Fin de Grado

Implementación y diseño de un configurador de recursos E2E para redes deterministas basadas en DetNet

---

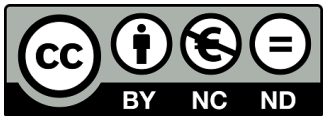
Alberto Martínez Méndez

Tutor/es

David Rico Menéndez

Antonio de la Oliva Delgado

17 de Junio de 2025 en Leganés



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

Este Trabajo de Fin de Grado aborda el diseño e implementación de un sistema modular para la provisión automática de servicios Extremo a extremo (E2E) sobre Deterministic Networking (DetNet), un paradigma de red emergente que garantiza comunicaciones con latencias acotadas y alta fiabilidad. La solución desarrollada integra múltiples componentes: un algoritmo de selección de ruta basado en restricciones de latencia y ancho de banda, mecanismos de configuración y desconfiguración remota de nodos mediante archivos JSON, un sistema de trazabilidad mediante registros (*logging*) y un módulo *watchdog* que monitoriza continuamente el sistema en busca de nuevos servicios.

El diseño ha sido validado mediante escenarios de prueba tanto controlados como aleatorios, incluyendo simulaciones de carga con grafos generados según el modelo de Erdős–Rényi. A través de estas pruebas se ha evaluado el correcto funcionamiento del sistema y el rendimiento computacional del algoritmo en función del número de nodos y la conectividad de la red. Los resultados obtenidos confirman la viabilidad y escalabilidad de la propuesta, así como su capacidad para adaptarse a entornos sensibles al tiempo como la automatización industrial, la robótica avanzada o los sistemas de transporte inteligente. La arquitectura planteada sienta las bases para futuras ampliaciones que integren técnicas más sofisticadas de optimización o inteligencia artificial.

Palabras clave: Redes deterministas, DetNet, configuración distribuida, algoritmo de selección de ruta, latencia, tráfico sensible al tiempo, servicios E2E.

SUMMARY

This Bachelor's Thesis addresses the design and implementation of a modular system for the automatic provisioning of E2E services over DetNet, an emerging networking paradigm that provides communications with bounded latency and high reliability. The proposed solution integrates several components: a route-selection algorithm constrained by latency and bandwidth, remote (de)configuration of nodes via JSON files, a traceability system based on *logging*, and a *watchdog* module that continuously monitors the system for new services.

The design has been validated through both controlled and random test scenarios, including load simulations on graphs generated with the Erdős–Rényi model. These experiments evaluate the system's functional correctness and the algorithm's computational performance as the number of nodes and network connectivity grow. The results confirm the feasibility and scalability of the proposal and its suitability for time-sensitive environments such as industrial automation, advanced robotics, or intelligent transportation systems. The presented architecture lays the groundwork for future extensions that incorporate more sophisticated optimisation techniques or artificial intelligence.

Keywords: Deterministic networks, DetNet, distributed configuration, route-selection algorithm, latency, time-sensitive traffic, E2E services.

DEDICATORIA

A mi familia, por su amor incondicional y por enseñarme desde siempre el valor del esfuerzo y la constancia. Gracias por acompañarme en cada paso, incluso en los más silenciosos. Sin vuestro apoyo, este camino no habría sido posible.

A Cristina, por ser mi refugio, mi impulso y mi calma. Gracias por creer en mí incluso cuando yo dudaba, por estar presente en los días difíciles y por celebrar conmigo cada pequeño logro.

Y a mi tutor de TFG, David, por su cercanía, orientación constante y por guiarme con paciencia y claridad en este proceso. Su confianza ha sido clave para que este proyecto llegara a buen puerto.

A todos vosotros, gracias por estar.

ÍNDICE GENERAL

1. INTRODUCCIÓN. 1
2. ESTADO DEL ARTE 3
2.1. Redes Deterministas (DetNet) 3
2.1.1. Introducción a DetNet 3
2.1.2. Características principales 4
2.1.3. Arquitectura DetNet 9
2.1.4. Importancia de las redes deterministas (DetNet) 11
2.1.5. Aplicaciones de las redes deterministas (DetNet) 11
2.2. Contexto 13
2.2.1. Marco PREDICT-6G. 13
2.2.2. Encaje del *E2E Configurator* 13
2.3. Tecnologías y soluciones existentes 13
2.3.1. Soluciones de Configuración y Orquestación en Redes Deterministas (DetNet) 13
2.3.2. Algoritmos de Selección de Rutas Orientados a QoS en Entornos Deterministas 14
2.3.3. Brechas Actuales y Desafíos Pendientes 17
3. ANÁLISIS Y ESPECIFICACIÓN DEL PROBLEMA 19
3.1. Descripción del servicio a configurar. 19
3.1.1. Identificación y Control del Servicio 19
3.1.2. Características de Calidad de Servicio (QoS) 19
3.1.3. Características del Tráfico 20
3.2. Requisitos del sistema 20
3.2.1. Requisitos Funcionales 20
3.2.2. Requisitos No Funcionales 21
3.3. Desafíos teóricos 22
3.4. Desafíos Técnicos del desarrollo 22
3.4.1. Procesamiento Eficiente de Peticiones JSON 23
3.4.2. Selección de rutas bajo restricciones de QoS 23

3.4.3. Gestión y control de recursos en la red . . . 23
3.4.4. Interacción con los Routers y Configuración de la Red . . . 23
3.4.5. Rendimiento, concurrencia y escalabilidad . . . 24
3.4.6. Seguridad en la configuración . . . 24
3.4.7. Supervisión del sistema y análisis del rendimiento . . . 24
4. DISEÑO DE LA SOLUCIÓN . . . 26
4.1. Arquitectura del Sistema . . . 26
4.1.1. Visión General . . . 26
4.1.2. Componentes Principales del Sistema . . . 26
4.2. Lógica del Procesamiento de Peticiones . . . 28
4.2.1. Recepción de la solicitud y validación . . . 28
4.2.2. Flujo de procesamiento y registro del servicio . . . 28
4.2.3. Diagrama de flujo del sistema . . . 30
4.3. Diseño del Algoritmo de Selección de Ruta . . . 33
4.3.1. Planteamiento del problema y requisitos . . . 33
4.3.2. Criterio de selección aplicado . . . 33
4.3.3. Diagrama de flujo del algoritmo . . . 34
4.3.4. Alternativas evaluadas y justificación . . . 35
4.3.5. Propuesta de mejora y optimización . . . 37
4.4. Diseño de la configuración y deconfiguración . . . 37
4.5. Componentes Técnicos Adicionales . . . 39
4.5.1. Gestión de procesos concurrentes . . . 39
4.5.2. Manejo de señales y control de interrupciones . . . 40
4.5.3. Sistema watchdog y configuración remota . . . 40
4.5.4. Robustez y tolerancia a errores . . . 40
4.5.5. Aspectos de seguridad en la configuración de red . . . 41
5. IMPLEMENTACIÓN DEL SISTEMA . . . 43
5.1. Estructura del código y organización del proyecto . . . 43
5.1.1. Distribución de módulos y carpetas . . . 43
5.1.2. Consideraciones generales sobre el desarrollo en Python . . . 44

5.2. Implementación del procesamiento de peticiones 45
5.2.1. Recepción y almacenamiento del archivo JSON 45
5.2.2. Procesamiento preliminar y validación de datos. 45
5.2.3. Carga de topología y sincronización con base de datos 46
5.2.4. Lectura del archivo y validación del servicio 46
5.2.5. Determinación del tipo de operación: alta o descarte 47
5.3. Procedimiento de descarte y liberación de recursos 47
5.3.1. Liberación de recursos en la topología de red 47
5.3.2. Actualización de la base de datos 48
5.3.3. Desconfiguración remota de los nodos implicados 48
5.4. Ejecución del algoritmo de selección de ruta 48
5.4.1. Obtención de parámetros de QoS y datos de red 49
5.4.2. Evaluación de rutas viables y validación 49
5.4.3. Selección de la ruta óptima y actualización del sistema 50
5.5. Automatización mediante watchdog y configuración remota 51
5.5.1. Monitorización de la base de datos 51
5.5.2. Envío de configuraciones a los nodos mediante SSH/SCP 52
5.5.3. Lógica de modificación de archivos de configuración en los nodos 52
5.6. Registro de eventos y trazabilidad del sistema. 54
5.6.1. Sistema de logs 54
5.6.2. Mensajes relevantes y monitoreo de fallos 54
Conclusión del capítulo 54
6. RESULTADOS Y EVALUACIÓN. 56
6.1. Diseño de los escenarios de prueba 56
6.1.1. Objetivos de validación y metodología. 56
6.1.2. Prueba funcional completa del sistema. 57
6.1.3. Verificación de la configuración remota de nodos 60
6.1.4. Simulación de carga mediante grafos Erdős–Rényi 60
6.2. Análisis de rendimiento del algoritmo 61
6.2.1. Medición de tiempos de ejecución 61
6.2.2. Comparativa en función del número de nodos 62

6.2.3. Discusión sobre la escalabilidad y eficiencia computacional 63
6.2.4. Discusión de resultados y lecciones aprendidas 64
7. CONCLUSIONES 66
8. PLANIFICACIÓN 67
9. MARCO REGULADOR 68
9.1. Legislación aplicable. 68
9.2. Normativa técnica 68
9.3. Ética y responsabilidad 69
9.4. Propiedad intelectual. 69
10. ENTORNO SOCIO-ECONÓMICO 70
10.1. Presupuesto de elaboración del TFG 70
10.2. Impacto socio-económico esperado 70
10.2.1. Impacto económico e industrial 70
10.2.2. Impacto social y de accesibilidad 71
10.2.3. Sostenibilidad y medio ambiente 72
10.2.4. Ética y responsabilidad. 72
10.3. Conclusión 72
11. TRABAJOS FUTUROS. 73
BIBLIOGRAFÍA 76

ÍNDICE DE FIGURAS

2.1 PREOF-Capable DetNet IP Domain. Adaptado de [10]. 6

2.2 Puerto de salida 1 de un *TSN bridge* con ventanas TAS. Fuente: [20]. . . . 8

2.3 Diagrama de *scheduling* TAS para el mismo puerto. Fuente: [20]. 8

2.4 Ejemplo de red habilitada con DetNet. Fuente: Figura 3 de [2]. 9

4.1 Diagrama de flujo del controlador principal (main.py) 30

4.2 Procesamiento de la petición HTTP en Flask mediante `upload_json()` . 30

4.3 Monitorización del directorio `input_files` y detección de nuevos archivos 31

4.4 Inicio del procesamiento del archivo JSON detectado 31

4.5 Flujo de procesamiento de un nuevo servicio a partir del JSON recibido . 32

4.6 Flujo de eliminación de un servicio ya activo (descartado) 32

4.7 Diagrama de flujo del algoritmo principal de selección de ruta. 35

4.8 Diagrama de flujo del subprocedimiento de validación de caminos (‘validate_path‘). 35

4.9 Esquema conceptual de una ruta DetNet con tres nodos intermedios (A, B, C) entre configuradores. 38

6.1 Estado inicial de la red: topología con 5 nodos, enlaces simétricos. 57

6.2 Primer servicio asignado: reducción de capacidad en los enlaces utilizados. 58

6.3 Red sin caminos viables: enlaces sin capacidad suficiente. 59

6.4 Recuperación de capacidad tras descartar el servicio con ID 2. 59

6.5 Tiempo de ejecución del algoritmo en función del número de nodos (p = 0.3). 62

6.6 Tiempo de ejecución del algoritmo en función del número de nodos (p = 0.2). 63

6.7 Tiempo de ejecución del algoritmo en función de la probabilidad de enlace (n = 10). 64

8.1 Diagrama de Gantt del proyecto 67

ÍNDICE DE TABLAS

4.1 Acciones configuradas en los nodos según su posición en el camino E2E. 39

5.1 Ejemplo de entrada en la base de datos `database.csv` 51

6.1 Estado de la base de datos tras el primer servicio 58

6.2 Estado de la base de datos tras dos asignaciones 59

6.3 Estado de la base de datos tras descarte 60

1. INTRODUCCIÓN

El avance de las tecnologías de red en entornos críticos ha impulsado el desarrollo de nuevas arquitecturas que permitan garantizar prestaciones de calidad de servicio extremadamente exigentes. En este contexto, las *Time-Sensitive Networking (TSN)* [1] y las redes deterministas (DetNet) [2] han surgido como una respuesta técnica frente a las necesidades de transmisión ultra confiable, con baja latencia y alta disponibilidad, propias de aplicaciones industriales, automoción, robótica o sistemas de control distribuido.

Sin embargo, el despliegue y configuración de servicios extremo a extremo (*end-to-end*) sobre este tipo de infraestructuras presenta múltiples desafíos. La reserva de recursos, la elección de rutas viables, la programación de nodos intermedios y la adaptación dinámica a nuevas solicitudes requieren no solo un conocimiento detallado de la topología de red, sino también mecanismos automatizados que permitan gestionar esta complejidad sin intervención humana constante.

El presente Trabajo de Fin de Grado propone una solución modular, programada en Python, capaz de automatizar la configuración de servicios E2E en una red determinista. A partir de una arquitectura distribuida basada en monitorización de archivos, lógica de selección de ruta y configuración remota mediante Secure Shell (SSH)/Secure Copy Protocol (SCP), el sistema diseñado permite recibir solicitudes en formato JSON, procesarlas de forma autónoma, calcular rutas óptimas y aplicar la configuración correspondiente en cada nodo de la red.

La solución aborda tanto el diseño de los componentes principales como su implementación e integración. Se incluyen pruebas funcionales completas, simulaciones de carga para analizar el rendimiento del algoritmo de selección, y se discuten los resultados obtenidos desde una perspectiva de escalabilidad y eficiencia computacional.

Además, se plantea una propuesta de mejora orientada a evolucionar el sistema hacia un modelo más inteligente, escalable y adaptativo, abriendo la puerta a futuras líneas de investigación como el uso de inteligencia artificial para la toma de decisiones de enrutamiento o la integración real con routers de red.

La estructura del documento es la siguiente:

- **Capítulo 1. Introducción:** contextualiza la relevancia de las redes deterministas, delimita los objetivos del TFG y describe la metodología seguida.
- **Capítulo 2. Estado del arte:** revisa los fundamentos de DetNet y TSN, el marco PREDICT-6G y las soluciones actuales de orquestación, algoritmos de ruta y brechas abiertas.
- **Capítulo 3. Análisis y especificación del problema:** detalla la petición de servicio,

los requisitos funcionales/no funcionales, los desafíos teóricos y técnicos que guían el diseño.

- **Capítulo 4. Diseño de la solución:** presenta la arquitectura del sistema, la lógica de procesamiento de peticiones, el algoritmo de selección de ruta y los componentes técnicos adicionales (watchdog, seguridad, etc.).
- **Capítulo 5. Implementación del sistema:** describe la estructura del código, la interacción entre módulos y la aplicación práctica de la configuración y desconfiguración remota de nodos.
- **Capítulo 6. Resultados y evaluación:** valida la solución mediante pruebas funcionales y de carga, midiendo tiempo de ejecución, escalabilidad y eficiencia computacional.
- **Capítulo 7. Conclusiones:** resume los logros, limitaciones y aporta una visión crítica de los resultados.
- **Capítulo 8. Planificación:** incluye el cronograma de trabajo, la distribución temporal de tareas y el diagrama de Gantt del proyecto.
- **Capítulo 9. Marco regulador:** analiza la legislación, normativa técnica, aspectos éticos y propiedad intelectual aplicables al desarrollo del configurador E2E.
- **Capítulo 10. Entorno socio-económico:** estima el presupuesto del TFG y evalúa el impacto económico, social, medioambiental y ético de la solución propuesta.
- **Capítulo 11. Trabajos futuros:** propone mejoras y líneas de investigación para evolucionar el algoritmo, integrar IA o ampliar la compatibilidad con otros dominios tecnológicos.

Con este enfoque, se pretende demostrar no solo la viabilidad técnica del sistema propuesto, sino también su potencial como base para futuras soluciones en el ámbito de la automatización de servicios deterministas en redes modernas.

2. ESTADO DEL ARTE

2.1. Redes Deterministas (DetNet)

2.1.1. Introducción a DetNet

Las redes DetNet [2] representan una evolución fundamental en el campo de las telecomunicaciones [3]. A diferencia del servicio *best-effort* de las redes tradicionales [4], el cual es suficiente para aplicaciones no críticas como la navegación web, donde las variaciones de retardo apenas afectan al usuario, DetNet reserva recursos y coordina la transmisión para garantizar que cada paquete llegue a destino dentro de un marco temporal acotado y con pérdidas mínimas. Estas prestaciones resultan cruciales en escenarios con requisitos estrictos de latencia y fiabilidad, tales como la automatización industrial, la conducción autónoma o la robótica médica, donde un retraso inesperado o la pérdida de un único paquete puede tener consecuencias significativas.

DetNet fue estandarizado por el Grupo de Trabajo de Redes Deterministas (Deterministic Networking Working Group) del Internet Engineering Task Force (IETF)[5] en colaboración con el grupo de trabajo IEEE 802.1 TSN [1]. Esta colaboración busca establecer una arquitectura común que permita ofrecer servicios deterministas tanto a nivel de enlace de datos (Capa 2 del modelo OSI)[6] como a nivel de red (Capa 3). Mientras que TSN opera en la capa de enlace de datos del modelo OSI para ofrecer tráfico determinista en redes Ethernet locales, DetNet extiende estas garantías a redes de área amplia IP/MPLS de varios saltos, de modo que los paquetes mantengan latencia y pérdida acotadas a lo largo de toda la ruta [2].

Al ejecutarse sobre redes IP abiertas, DetNet introduce riesgos de ciberseguridad. El RFC 9055 [7] identifica, entre otras amenazas, los **ataques de retardo**, en los que un adversario introduce demoras adicionales en el plano de datos o de control para violar los límites temporales del flujo y los **ataques de modificación o suplantación del flujo**, donde paquetes falsificados alteran la temporización o el contenido del tráfico determinista. Estos escenarios, junto con los ataques **man-in-the-middle** clásicos, obligan a aplicar medidas como autenticación de nodos, protección de la integridad de los paquetes y redundancia de rutas para preservar las propiedades deterministas del servicio.

El concepto de redes deterministas tiene sus raíces en los esfuerzos iniciales de la IEEE 802.1 para desarrollar Audio Video Bridging (AVB) en 2005 [8], que permitía la transmisión de flujos de audio y video en tiempo real. Sin embargo, AVB no toleraba fallos y tenía unas grandes limitaciones en cuanto a seguridad y fiabilidad. Esto se solucionó en 2012 con la introducción del concepto TSN, el cual mejoró la sincronización en tiempo.

Aunque TSN podía cubrir las necesidades en entornos LAN [1], estas redes no podían

extenderse a redes WAN de manera eficiente [9]. Por este motivo, en 2015 el IETF, en colaboración con diversas organizaciones de desarrollo de estándares (SDOs) y el IEEE 802, estableció el Grupo de Trabajo DetNet [5]. El objetivo principal de DetNet es proporcionar una conectividad determinista de E2E a través de redes IP utilizando técnicas avanzadas de reserva de recursos y encaminamiento explícito para garantizar tiempos de latencia predecibles, baja pérdida de paquetes y alta fiabilidad [2].

En conclusión, DetNet representa un paso adelante hacia la creación de redes que puedan garantizar servicios deterministas en redes Wide Area Network (WAN). Su colaboración con TSN y la adaptación de técnicas de reserva de recursos, replicación de paquetes y eliminación de duplicados permitirán a las industrias implementar aplicaciones críticas con requisitos estrictos de latencia y fiabilidad, lo que las hará indispensables para el futuro de las redes sensibles al tiempo.

2.1.2. Características principales

Las redes deterministas (DetNet) se distinguen por una serie de características que las hacen aptas para aplicaciones que requieren una alta fiabilidad, previsibilidad y control estricto de los recursos.

Tal como se ha mencionado anteriormente, en aplicaciones donde la transmisión de datos debe cumplir con requisitos estrictos de sincronización y fiabilidad como sistemas de control en entornos industriales, redes críticas de comunicación o plataformas de monitorización en tiempo real, resulta fundamental implementar un control riguroso sobre cuándo y cómo se envían los datos.

A continuación, se describen las principales características que hacen que DetNet sea una opción ideal para estos entornos sensibles al tiempo.

Baja Latencia Garantizada

Una de las características más destacadas de las redes deterministas es la garantía de baja latencia. En los entornos DetNet, los paquetes se entregan dentro de un marco de tiempo prestablecido, lo que elimina la incertidumbre en tiempo que puede causar la congestión de la red o el comportamiento aleatorio de las rutas de transmisión.

Para conseguir esta reducción de latencia y conseguir que el paquete sea recibido en el marco de tiempo prestablecido, se deben de asignar recursos dedicados a cada flujo de datos a través de técnicas de planificación y reserva de recursos que eviten las congestiones y los cuellos de botella.

¿Por qué es importante la garantía de baja latencia? La latencia es crucial en aplicaciones donde los tiempos de respuesta son estrictos, críticos y predecibles para garantizar un funcionamiento seguro y eficiente. En estos casos, incluso retrasos de milisegundos pueden tener consecuencias graves. Por ejemplo, en los sistemas de conducción

de vehículos autónomos, la comunicación entre sensores, cámaras y unidades de procesamiento requiere una latencia extremadamente baja para que el vehículo pueda tomar decisiones inmediatas, como frenar ante un obstáculo inesperado o evitar colisiones.

La tecnología DetNet busca ofrecer garantías determinísticas de baja latencia mediante el uso de rutas predefinidas, control de la congestión y sincronización precisa entre nodos. Esto permite que las redes puedan soportar aplicaciones sensibles al tiempo (Time-Sensitive Applications) y asegurar el cumplimiento de los requisitos de latencia establecidos, evitando fluctuaciones o retrasos imprevistos que podrían comprometer la integridad de los servicios críticos.

Redundancia y Recuperación Rápida

La fiabilidad es uno de los pilares fundamentales de las redes deterministas, y uno de los mecanismos clave para garantizarla es la implementación de redundancia y recuperación rápida. En los entornos DetNet, la posibilidad de fallos en las rutas de transmisión no debe afectar la continuidad del servicio, lo cual es vital en aplicaciones críticas. Para lograr este objetivo, DetNet implementa mecanismos de protección contra posibles fallos como la replicación de paquetes y el uso de rutas prestablecidas. Por tanto en caso de que algún enlace o nodo intermedio falle, los paquetes de datos siguen llegando a su destino correctamente sin interrupciones ni intervención manual.

Para comprender mejor la importancia de los mecanismos de redundancia en redes DetNet, retomemos el caso de los vehículos autónomos. Si bien la baja latencia es esencial para que el sistema pueda reaccionar a tiempo ante situaciones críticas, no basta con asegurar únicamente la rapidez en la transmisión de datos. En un entorno donde decisiones en milisegundos pueden marcar la diferencia, la pérdida de paquetes clave como la posición de un peatón o la distancia a otro vehículo, podría comprometer gravemente la seguridad. Por ello, además de minimizar los retrasos, DetNet implementa estrategias de replicación y recuperación rápida que garantizan la entrega fiable de los datos, incluso ante fallos en las rutas de transmisión.

Uno de los mecanismos técnicos más relevantes para asegurar esta protección es el conjunto de funciones conocido como Packet Replication, Elimination, and Ordering Functions (PREOF) [2]. Este mecanismo no solo permite la creación de rutas redundantes, sino que también gestiona el proceso de replicación, eliminación de duplicados y reordenación de paquetes para garantizar un flujo de datos coherente y fiable, incluso en condiciones de fallo.

¿En qué consiste PREOF?

PREOF, como su propio nombre indica, es un conjunto de funciones definidas en la subcapa de servicio de DetNet para proporcionar protección de servicio en redes deterministas.

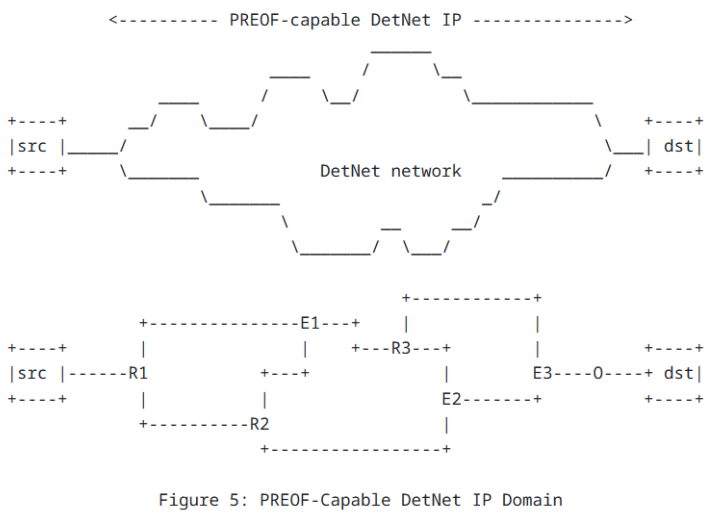


Fig. 2.1. PREOF-Capable DetNet IP Domain. Adaptado de [10].

El mecanismo se divide en tres funciones principales que trabajan conjuntamente para asegurar la entrega fiable de paquetes:

1. **Packet Replication (Replicación de paquetes)**: Consiste en generar copias de los mismos paquetes y enviarlas por múltiples rutas disjuntas.
2. **Packet Elimination (PEF)**: Consiste en eliminar duplicados basándose en la información de secuenciación.
3. **Packet Ordering (POF)**: Consiste en restaurar el orden correcto de los paquetes en caso de que lleguen fuera de secuencia.

Mecanismos de fiabilidad: PREOF y FRER

El mecanismo PREOF (Packet Replication, Elimination and Ordering Functions) es una subcapa de servicio de DetNet que replica, elimina duplicados y reordena paquetes, garantizando la entrega fiable de los datos [2]. PREOF es independiente del plano de datos y puede aplicarse sobre MPLS, IP o MPLS-sobre-UDP/IP [11]-[13]; la última opción facilita la interoperabilidad con infraestructuras IP existentes [10].

A nivel 2, las redes TSN disponen del estándar **FRER** (Frame Replication and Elimination for Reliability), definido en IEEE 802.1CB, que replica tramas en enlaces disjuntos dentro de un dominio Ethernet y elimina duplicados al llegar al destino [14]. Un dominio *DetNet Edge* puede, por tanto, emplear PREOF entre nodos DetNet para la protección extremo a extremo mientras utiliza FRER en los subredes TSN locales que atraviesa. El controlador DetNet orquesta ambos mecanismos: configura FRER en los puentes de capa 2 y, a la vez, establece los flujos PREOF en el plano de datos IP/MPLS, combinando fiabilidad intra-dominio y protección E2E.

Sincronización Precisa

La sincronización precisa es uno de los pilares esenciales para el correcto funcionamiento de las redes deterministas, ya que permite que todos los dispositivos en la red operen bajo una referencia temporal común. En entornos donde la coordinación de múltiples sistemas en tiempo real es crítica, una mínima desincronización podría provocar fallos de comunicación o latencias inesperadas.

Para garantizar esta sincronización, DetNet se basa en protocolos de sincronización de tiempo como Precision Time Protocol (PTP) [15]. Este protocolo permite que los dispositivos de la red puedan sincronizar sus relojes internos con una precisión que puede llegar hasta el nivel de nanosegundos, lo cual es esencial en aplicaciones donde se usa TSN.

La sincronización precisa es un requisito indispensable en DetNet, ya que garantiza que todos los dispositivos de la red operen de manera coordinada en entornos donde los tiempos de respuesta estrictos son críticos. Sin esta sincronización, las redes deterministas no podrían cumplir con los requisitos de baja latencia, previsibilidad y fiabilidad, comprometiendo la integridad de los servicios críticos.

Previsibilidad y Control Estricto de Recursos

Como ya hemos comentado anteriormente, la previsibilidad es una de las características mas importantes de DetNet debido a que, si conseguimos que cada flujo de datos se entregue de forma consistente y dentro de los límites de tiempo previamente establecidos, sin depender de las condiciones variables del tráfico de la red, podremos reducir considerablemente la latencia, la perdida de paquetes y la variabilidad en los tiempos de entrega (jitter).

Para lograr esta previsibilidad, DetNet trata cada flujo de datos (data flow) como una entidad independiente, las cuales tiene cada una unos requisitos específicos que deben de ser atendidos por técnicas de planificación y control de recursos estrictos. Por ejemplo la asignación del ancho de banda, la gestión de colas de trasmisión (Mecanismos de scheduling) y el uso de rutas predefinidas.

Una de las claves para mantener este control estricto es la capacidad de Traffic Engineering (TE) [16], que permite planificar rutas óptimas y evitar congestiones en la red. Además, DetNet puede emplear protocolos como RSVP (Resource Reservation Protocol) [17] para la reserva explícita de recursos a lo largo de toda la ruta de transmisión. Esto asegura que los flujos críticos tengan siempre los recursos necesarios para cumplir con los requisitos de latencia y fiabilidad, incluso en situaciones de alta demanda.

Uno de los mecanismos más relevantes *en el dominio de capa 2 TSN* es el **Time-Aware Shaper (TAS)** [18], estandarizado en IEEE 802.1Qbv [19]. TAS abre y cierra las puertas de los búferes de salida siguiendo un *Gate Control List* (GCL), de modo que los paquetes críticos no colisionen con otros flujos. Figure 2.2 y Figure 2.3, tomadas de la

tesis de Larrañaga Zumeta [20], muestran su funcionamiento en un *TSN bridge*.

DetNet opera en capa 3 y no utiliza directamente 802.1Qbv; sin embargo, *puede configurar* dominios TSN subyacentes para aportar protección temporal en la LAN. Para el tránsito *entre* dominios IP/MPLS, DetNet adopta políticas de *Traffic Engineering* orientadas a latencia —por ejemplo, *Cyclic Queuing and Forwarding (CQF)* recogidas en RFC 9320 [21]— que se inspiran en el principio de ventanas temporales sin ser TAS. Así, un flujo extremo-a-extremo puede beneficiarse de TAS en la red de acceso y de CQF/DetNet en el *core*.

Para que este *scheduling* funcione, nodos finales y *bridges* deben sincronizarse según IEEE 802.1AS [20], garantizando la apertura/cierre de puertas en el instante preciso.

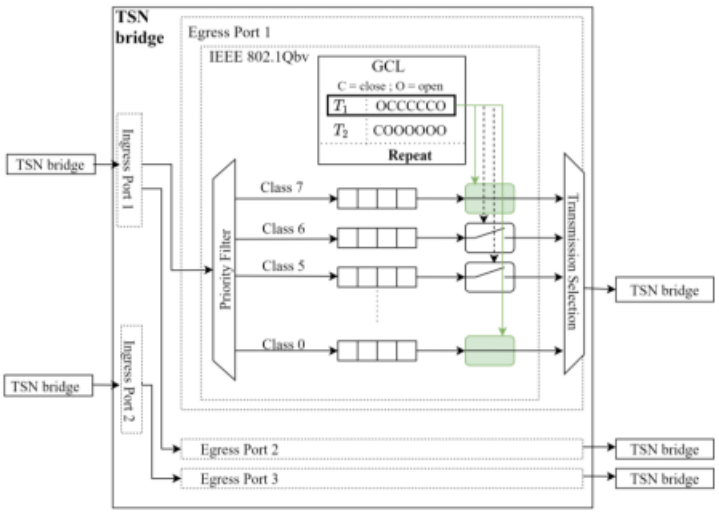


Fig. 2.2. Puerto de salida 1 de un *TSN bridge* con ventanas TAS. Fuente: [20].

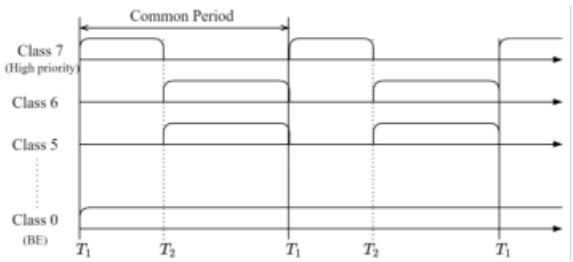


Fig. 2.3. Diagrama de *scheduling* TAS para el mismo puerto. Fuente: [20].

Para admitir este tipo de *scheduling*, todos los nodos, tanto los usuarios finales como los *bridges*, deben estar sincronizados utilizando un reloj maestro y varios relojes esclavos, como se define en el estándar IEEE 802.1AS [20].

2.1.3. Arquitectura DetNet

La arquitectura de redes deterministas definida por el IETF (DetNet) tiene como objetivo proporcionar una conectividad fiable, de baja latencia y con pérdida mínima de paquetes en redes IP y/o MPLS. Esta arquitectura está especialmente diseñada para ofrecer un comportamiento determinista de E2E, incluso en redes de múltiples saltos, replicando así las capacidades de TSN en redes de área amplia.

La arquitectura DetNet se aplica típicamente dentro de un dominio controlado por una única entidad administrativa, como una red privada o de campus. A través de técnicas como la reserva de recursos, rutas explícitas y redundancia en el envío, DetNet es capaz de garantizar que los flujos de datos sensibles al tiempo cumplan requisitos estrictos de calidad de servicio.

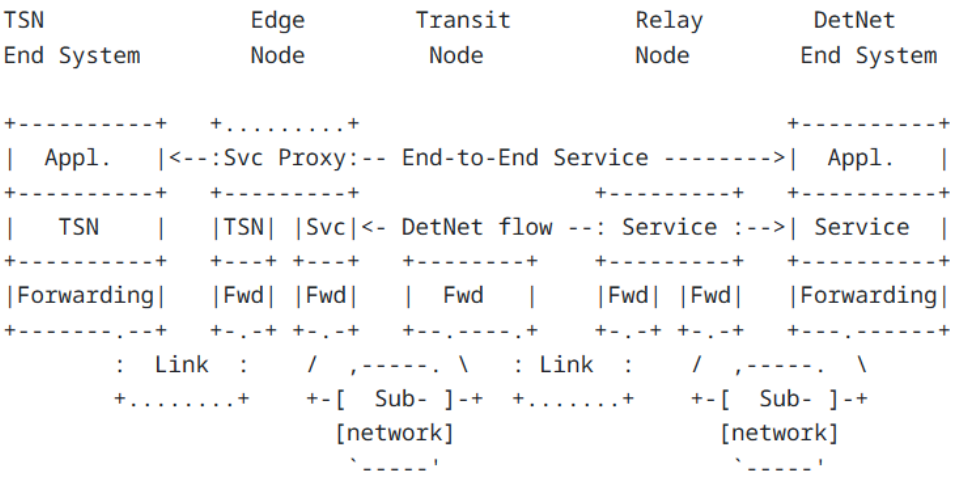


Fig. 2.4. Ejemplo de red habilitada con DetNet. Fuente: Figura 3 de [2].

Como puede observarse en la Figura 2.4, la arquitectura DetNet distingue varios tipos de nodos que forman parte del camino de un flujo determinista:

- **Edge Node (nodo de borde)**: se encarga de clasificar los paquetes entrantes de una aplicación, mapearlos a un flujo DetNet específico y aplicar la encapsulación necesaria. También puede implementar funciones PREOF (replicación y eliminación de paquetes).
- **Transit Node (nodo de tránsito)**: participa únicamente en el reenvío del flujo, respetando las reservas de recursos establecidas, sin aplicar funciones de servicio adicionales.
- **Relay Node (nodo de retransmisión)**: combina las funciones de reenvío y servicio, ejecutando PREOF en puntos intermedios para aumentar la resiliencia del flujo.
- **End Systems**: representan los sistemas que envían y reciben los flujos deterministas.

Plano de datos

El plano de datos DetNet se estructura en dos subcapas bien diferenciadas:

- **Subcapa de forwarding**: se encarga de transportar los paquetes por la red utilizando conmutación IP o MPLS, pero con reservas específicas por flujo (como ancho de banda dedicado, buffers y colas de prioridad). Los nodos de tránsito operan principalmente en esta subcapa.
- **Subcapa de servicio**: es responsable de aplicar técnicas de replicación, eliminación y ordenamiento de paquetes (PREOF) para garantizar una entrega fiable incluso en condiciones de fallo. Se implementa en nodos de borde o retransmisión.

Gracias a esta división, la arquitectura puede garantizar que los flujos con requisitos deterministas se encaminen con recursos reservados, aplicando protección contra fallos y evitando congestión o pérdidas no deseadas.

Plano de control y plano de gestión

DetNet también contempla un plano de control y un plano de gestión (que pueden combinarse en un plano de controlador) responsable de establecer las rutas deterministas y reservar los recursos adecuados en cada nodo. Este plano puede implementarse con:

- Protocolos distribuidos tradicionales extendidos, como RSVP-TE adaptado a DetNet [22].
- Controladores SDN centralizados que calculan rutas óptimas y configuran los nodos antes del inicio del flujo.

El plano de gestión, por su parte, se ocupa de tareas como la configuración inicial de la red, la monitorización del rendimiento (OAM), y la provisión de interfaces norte (northbound) que permiten a las aplicaciones solicitar servicios deterministas con requisitos explícitos (latencia máxima, ancho de banda mínimo, etc.).

En conjunto, la arquitectura DetNet permite diseñar flujos deterministas con garantías estrictas de calidad de servicio, gracias a la coordinación entre los distintos planos (datos, control y gestión) y a la colaboración entre nodos que implementan funciones diferenciadas. Esta segmentación funcional facilita la escalabilidad de DetNet y su integración con tecnologías existentes, permitiendo su uso tanto en redes locales como en redes de área amplia.

2.1.4. Importancia de las redes deterministas (DetNet)

En los últimos años estamos viendo un crecimiento de las llamadas çiudades inteligentes". Estamos viendo como las ciudades, los vehículos y las industrias se vuelven más autónomos y por tanto el papel de las redes deterministas se vuelve cada vez más importante. El hecho de que las redes deterministas como DetNet sean capaces de gestionar este tipo de necesidades, la convierten en unas redes muy necesarias para el funcionamiento de aplicaciones que requieren interacciones en tiempo real entre máquinas, dispositivos y sistemas distribuidos.

Uno de los aspectos más relevantes de DetNet es poder garantizar un servicio de E2E confiable. Esto no solo es importante para aplicaciones que envíen datos críticos, como los vehículos autónomos o las redes industriales, sino también para sectores que demandan una mayor precisión en la transmisión de datos. La capacidad de poder ofrecer las capacidades mencionadas anteriormente, hace que las redes deterministas se conviertan en una tecnología de suma importancia en la industria actual. En sectores donde la interrupción de la comunicación puede tener consecuencias graves, como en la atención sanitaria remota o los sistemas de control industrial, la fiabilidad se convierte en un factor indispensable.

Por tanto DetNet abre el camino para nuevas oportunidades en áreas como la automatización industrial, la robótica avanzada y la telemedicina, pudiendo cumplir los requisitos de estas aplicaciones.

2.1.5. Aplicaciones de las redes deterministas (DetNet)

Las redes deterministas (DetNet), al ofrecer garantías de latencia, fiabilidad, sincronización y previsibilidad, están convirtiéndose en la solución adoptada por diferentes industrias para sus retos tecnológicos. DetNet es clave en aplicaciones que requieren precisión y continuidad en la transmisión de información. En esta sección se describen algunas de las principales áreas de aplicación donde las redes deterministas están marcando la diferencia.

Automatización industrial

Uno de los sectores donde mas impactos esta teniendo DetNet es en la automatización industrial. Actualmente estamos sufriendo la Cuarta Revolución Industrial también llamada Industria 4.0. La industria 4.0 según la UNIR se define como .[El] proceso de transformación digital de las diferentes organizaciones industriales con la ayuda de las últimas tecnologías en el mercado relacionadas con el ecosistema 4.0, tales como el Big Data, el blockchain, la ciberseguridad, la robótica o la aplicación del Internet of Things (IoT)."[23] y esta suponiendo una serie de cambios y nuevas necesidades en la industria. Por ejemplo los sistemas de producción dependen de la sincronización precisa entre máquinas, senso-

res y actuadores y DetNet consigue cubrir estas necesidades de manera que los sistemas funcionen de manera confiable, asegurando que los datos lleguen en el momento exacto necesario para ejecutar tareas críticas.

En este sector la fiabilidad es crucial para evitar interrupciones en la producción y prevenir errores. Por ejemplo, un sistema robótico avanzado que depende de la transmisión de datos en tiempo real no puede tolerar retrasos o pérdida de información, ya que esto podría provocar errores de precisión o fallos operativos. Por tanto DetNet consigue reducir el impacto industrial y económico que puedan provocar los retrasos o las perdidas.

Vehículos autónomos y transporte conectado

El sector de la automoción está experimentando una transformación radical con el desarrollo de vehículos autónomos y sistemas de transporte inteligente (ITS). Este tipo de vehículos pueden hacer uso de redes deterministas DetNet por ejemplo para la gestión de grandes volúmenes de datos de tráfico en tiempo real.

La comunicación entre vehículos (Vehicle-to-Vehicle, V2V) y entre vehículos e infraestructuras (Vehicle-to-Infrastructure, V2I) requiere tiempos de respuesta extremadamente bajos y una alta fiabilidad para poder evitar accidentes y poder tomar decisiones críticas en milisegundos incluso superando la capacidad de reacción humana. Para ello se puede combinar con tecnologías como 5G o 6G para reducir al máximo los tiempos de respuesta mientras que DetNet se encargaría de la alta fiabilidad.

Robótica médica y telemedicina

En el ámbito sanitario, las aplicaciones que hacen uso de las redes deterministas están sumando relevancia en áreas como la robótica médica o la cirugía a distancia.

La realización de cirugía a distancia consiste en la ejecución de procedimientos quirúrgicos por parte de un cirujano que no se encuentra físicamente presente en el lugar donde está el paciente, utilizando tecnologías avanzadas de comunicación y robótica para controlar instrumentos quirúrgicos de forma remota con alta precisión y en tiempo real.

DetNet asegura que los movimientos del robot se realicen en un marco de tiempo prestablecido y sin errores, reduciendo el riesgo de fallos durante el procedimiento quirúrgico. Además en este caso concreto las consecuencias de tener un fallo ya sea de alta latencia o de pérdida de paquetes son muy graves, por tanto necesitamos que la red sea muy efectiva y fiable.

La capacidad de DetNet para proporcionar conectividad determinista en estos casos puede marcar la diferencia entre la vida y la muerte, especialmente en situaciones donde los médicos necesitan tomar decisiones basadas en datos de monitorización que deben llegar de manera inmediata y sin interrupciones.

2.2. Contexto

2.2.1. Marco PREDICT-6G

PREDICT-6G propone una arquitectura 6G orientada a servicios deterministas *end-to-end* (Principio 1) sobre infraestructuras multi-tecnología y multi-dominio (Principio 2). El control se organiza en un plano AI-driven *Inter-domain Control-Plane* (AICP) y un plano de usuario *Multi-domain Data Plane* (MDP), tal como se resume en la Fig. 1 del artículo [24]. Cada dominio tecnológico (TSN Ethernet, Wi-Fi-TSN, 3GPP TSC, DetNet IP/MPLS) expone sus capacidades mediante **Management Services** (MS) específicos, mientras que un único *E2E MS* compone y asegura el servicio determinista a lo largo de todos ellos.

2.2.2. Encaje del *E2E Configurator*

El módulo `Configurator E2E` desarrollado en este TFG se integra como un **Management Service de Configuración** dentro del E2E MS de la AICP:

- **Entrada:** recibe la petición JSON procedente del *Service Automation MS* (§ 4.2).
- **Procesado:** calcula la ruta determinista y asigna recursos (PREOF/FRER, ancho de banda, GCL) empleando la topología expuesta por el *Path Computation MS*.
- **Salida:** envía la configuración a los *Resource Configuration MS* de cada dominio (TSN, DetNet, 3GPP), siguiendo las APIs abiertas modelo-driven recomendadas por PREDICT-6G.

Gracias a la *modularidad* (Principio 3) de la AICP, el algoritmo de ruta y reserva puede evolucionar independientemente del resto de servicios (véase la propuesta de mejora en § 4.3.5); bastará con sustituir el módulo sin impacto en los MS adyacentes.

En resumen, el `Configurator E2E` actúa como puente entre la solicitud de servicio y la configuración distribuida de los dominios, materializando los principios de multi-dominio, model-driven y automatización defendidos por PREDICT-6G [24].

2.3. Tecnologías y soluciones existentes

2.3.1. Soluciones de Configuración y Orquestación en Redes Deterministas (DetNet)

Las redes deterministas, como Time-Sensitive Networking (TSN) [1] en Ethernet, requieren mecanismos de configuración especializados para garantizar latencia acotada, baja fluctuación (jitter) y alta fiabilidad. Los estándares TSN han madurado en el plano de datos, pero la gestión de la configuración de recursos sigue evolucionando. En 802.1Qcc

(2018) [25] se definieron tres modelos de gestión TSN (distribuido, centralizado-distribuido, y totalmente centralizado) pero sin especificar protocolos de control detallados, lo que dificulta una configuración unificada de dispositivos TSN de distintos fabricantes. Para abordar esta orquestación, se han explorado arquitecturas SDN (Software-Defined Networking) [26] [27]y modelos de datos estandarizados. Por ejemplo, controladores SDN genéricos como OpenDaylight [28] u ONOS [29] pueden actuar como gestores TSN aprovechando que sus interfaces sur (southbound) soportan NETCONF [30](protocolo de configuración de red) y usan modelos YANG [31] para definir la configuración. OpenDaylight, en particular, incorpora directamente modelos YANG en su estructura de datos interna, facilitando la programación de dispositivos TSN vía NETCONF. Esto permite integrar TSN en un esquema SDN centralizado, mejorando la programabilidad y eficiencia en la gestión de recursos temporizados. De hecho, estudios reportan que introducir un controlador SDN en TSN agiliza la configuración de flujos y la asignación de recursos de forma consistente en la red.

Otra línea tecnológica es el uso de herramientas de automatización y modelado para configurar elementos de red deterministas. Por un lado, se emplean modelos YANG específicos (por ejemplo, definidos en IEEE 802.1Qcp/Qcx [32], [33]) que describen parámetros TSN (como listas de puertas de tiempo, reservas de flujo, etc.), los cuales pueden ser manipulados mediante herramientas de automatización.

En el ámbito industrial, la convergencia de OPC UA (Open Platform Communications Unified Architecture) [34] y TSN (Time-Sensitive Networking) [35] está emergiendo como una solución clave para lograr comunicaciones deterministas y estandarizadas de extremo a extremo. OPC UA proporciona un modelo de información unificado y versátil, con dos modos de comunicación: cliente-servidor y publicación/suscripción (Pub/Sub) .

La integración de OPC UA con TSN [36] combina la interoperabilidad y la semántica rica de OPC UA con las capacidades de transmisión determinista de TSN. En el modo cliente-servidor, OPC UA puede utilizarse para configurar nodos TSN, permitiendo que un controlador industrial solicite conexiones con requisitos específicos de calidad de servicio (QoS) a través de mensajes OPC UA [37]. Por otro lado, el modo Pub/Sub de OPC UA facilita la transmisión eficiente de datos en tiempo real sobre redes TSN, siendo adecuado para aplicaciones como la sincronización de datos de sensores [38].

La literatura reciente destaca que la combinación de OPC UA y TSN es fundamental para la implementación de la Industria 4.0, ya que permite una comunicación interoperable y determinista desde el nivel de sensores hasta la nube [34].

2.3.2. Algoritmos de Selección de Rutas Orientados a QoS en Entornos Deterministas

El encaminamiento de flujos en redes deterministas debe considerar criterios de QoS estrictos, principalmente latencia end-to-end garantizada y suficiente ancho de banda, evi-

tando congestión y variaciones temporales. Se han adaptado o desarrollado diversos algoritmos clásicos para este contexto:

- **Dijkstra Adaptado:** Muchas implementaciones básicas emplean el algoritmo de Dijkstra [39] para encontrar caminos mínimos, asignando pesos a los enlaces en función de la métrica relevante (por ejemplo, el retardo propagation + queuing o una combinación de retraso e utilización). Dijkstra ofrece la ruta de menor costo sumatorio, pero en su forma estándar no incorpora restricciones de schedulability. En entornos TSN/DetNet típicamente se usa Dijkstra para obtener rutas candidatas, pero asumiendo que luego se verificará si pueden cumplir las ventanas de tiempo de transmisión. Estudios [40] y [41] señalan que es común utilizar Dijkstra en la selección de ruta TSN, aunque esto no captura la interacción con la programación temporal de cada flujo. Por ello, surgen versiones adaptadas que integran criterios adicionales en el costo: por ejemplo, penalizar enlaces casi saturados o con mayor delay, o limitar la búsqueda solo a rutas que puedan cumplir cierto plazo (deadline). Dijkstra adaptado sirve como base eficiente, pero puede requerir filtros posteriores para asegurar que el camino seleccionado soporta tráfico crítico sin violar sus cotas temporales.

- **K-Shortest Paths (K rutas más cortas):** En escenarios donde se necesita redundancia o múltiples opciones (p. ej. para protección 1+1, donde se encaminan flujos duplicados por caminos disjuntos [2]), los algoritmos de k-rutas más cortas son útiles. Generan no solo la ruta mínima, sino las siguientes mejores rutas (segundo mejor camino, tercero, etc.), típicamente mediante variantes de Dijkstra iterativas [42]. En redes deterministas, obtener varias rutas permite luego evaluar cuál cumple mejor los requisitos de QoS o reservar rutas alternativas para respaldo. Un ejemplo es la protección 1+1 en DetNet [43], donde se requieren dos caminos disjuntos de baja latencia para cada flujo crítico [2]. Otro caso representativo es el mecanismo PREOF (Packet Replication, Elimination, and Ordering Functions), que permite enviar paquetes replicados por múltiples rutas y eliminarlos en destino si han llegado duplicados, lo que mejora la fiabilidad sin reenvíos ni intervención adicional [10]. El enfoque típico es calcular el camino primario más corto y luego un segundo camino disjunto óptimo (lo que puede hacerse con algoritmos específicos como Suurballe o buscando k-shortest simple-disjoint). Estos algoritmos no garantizan por sí solos la planificabilidad temporal, pero reducen el espacio de búsqueda para posteriormente asignar horarios de transmisión. En suma, k-shortest paths ofrecen diversidad de rutas manteniendo cercanía al óptimo de longitud, lo cual es valioso para distribuir carga o implementar esquemas de multi-camino determinista.

- **Búsqueda A*:** El algoritmo A* es una extensión informada de Dijkstra que incorpora una heurística para guiar la exploración hacia el destino, reduciendo el esfuerzo computacional en redes grandes [44]. En redes deterministas, A* puede emplear como heurística una estimación del retardo restante hasta el destino (por

ejemplo, la distancia euclídea si se tienen coordenadas, o un cálculo subestimado del retraso mínimo posible). Esto permite hallar la ruta más rápida (menor latencia) más eficientemente que Dijkstra puro. Adaptaciones de A* han sido propuestas para entornos de red dinámicos o dependientes del tiempo, donde los tiempos de enlace varían: por ejemplo, Grajales et al. extienden A* para computar caminos más rápidos en redes con tiempos de viaje dependientes del momento, consiguiendo rutas óptimas en escenarios de latencia variable [45]. En redes deterministas con ventanas temporales (time slots), A* también puede integrar en la heurística información de sincronización o próximos instantes de ventana abierta en cada enlace. La ventaja de A* es que garantiza el óptimo (si la heurística es admisible) pero explorando menos nodos que Dijkstra, resultando útil cuando se requieren cálculos casi en tiempo real para admitir nuevas conexiones con plazo acotado[46].

- **Modelos ILP (Integer Linear Programming):** Varios trabajos han formulado la selección de rutas con QoS como un problema de optimización entera lineal, integrando también la programación de envíos en el tiempo [47], [48]. Estos modelos ILP buscan asignar a cada flujo un camino y (posiblemente) un calendario de transmisión en switches, minimizando la utilización de recursos o la tardanza y respetando restricciones de capacidad y retardo límite [48]. Un estudio pionero en TSN formuló conjuntamente el ruteo y la calendarización de tramas periódicas como ILP bajo el paradigma SDN centralizado [47]. La solución por ILP garantiza un resultado óptimo dada la entrada (lista de flujos y sus requisitos), cumpliendo estrictamente las cotas de latencia con uso eficiente de enlaces [47]. Sin embargo, resolver ILPs es *NP-hard* y su complejidad crece rápidamente con el número de nodos y flujos, volviéndose impracticable para cálculo en línea en redes grandes o dinámicas [47]. Un caso de estudio mostró que, aun obteniendo soluciones exactas, el modelo no contemplaba detalles prácticos como la inclusión de *guard bands* antes de las ventanas TSN, lo que degradaba las garantías de latencia [49]. En la práctica, los ILP se emplean sobre todo para planificación *off-line* o como referencia teórica para métodos heurísticos. Entre los enfoques alternativos destacan los modelos de Programación por Restricciones (SMT) que, de modo similar al ILP, permiten sintetizar de forma exacta horarios y rutas en redes tiempo-real multi-salto [50].

- **Heurísticas específicas:** Dada la dificultad de escalar los ILP, se han propuesto diversas heurísticas y meta-heurísticas para el ruteo y la planificación con QoS determinista:

  - *Búsqueda Tabú*: empleada para resolver conjuntamente rutas y horarios en TSN, con latencias cercanas al ILP en un tiempo de cómputo mucho menor [47], [51].
  - *Greedy incremental*: asigna rutas según el ancho de banda residual y ajusta los horarios al llegar nuevos flujos, evitando reaprovisionar la red completa [49].

- Algoritmos *genéticos* e híbridos con Evolución Diferencial que ofrecen soluciones a pocos puntos del óptimo en tiempos sustancialmente más bajos que el ILP [52].
- Heurísticas de *colonia de hormigas*, que reducen de forma significativa la latencia media frente a Dijkstra en escenarios de fábricas inteligentes [53].
- Enfoques de *SMT* y *Branch & Bound*, capaces de sintetizar horarios exactos en redes multi-salto con tiempos sub-segundo para decenas de flujos [50].

- **Simulación con Digital Twin:** Los gemelos digitales de red permiten ensayar fuera de línea las rutas y horarios TSN/DetNet antes de aplicar la configuración real, acelerando la búsqueda con IA/ML y reduciendo riesgos. Ben Hadj Said et al. presentan un *Network Digital Twin* que valida nuevas configuraciones de TSN en 6G y actúa como banco de pruebas para algoritmos de gestión [54]. En el ámbito industrial, se indica que TSN es la tecnología habilitadora para gemelos digitales de fábrica, al ofrecer datos deterministas y sincronizados en tiempo real [55].

La elección del algoritmo depende del caso de uso: en entornos con pocos flujos estáticos, un ILP puede ser viable; en entornos dinámicos de gran escala, heurísticas como Tabú, k-shortest con filtrado, o búsquedas A* informadas son más prácticas para garantizar QoS sin sacrificar demasiado rendimiento.

2.3.3. Brechas Actuales y Desafíos Pendientes

A pesar de los avances, la literatura identifica brechas que el configurador e2e propuesto pretende subsanar en redes deterministas [56], [57].

- **Automatización sin apoyo SDN completo:** La mayoría de propuestas parten de que existe un controlador SDN central potente [27]. En el borde o en plantas heredadas, ese plano de control puede ser inviable por falta de CPU, memoria o coste [58]. IEEE 802.1Qcc define tres esquemas de provisión —distribuido, centralizado-distribuido y totalmente centralizado—; cuando la topología crece, la coordinación punto-a-punto de los modelos no centralizados se vuelve difícil de mantener [59], [60]. Además, 802.1Qcc no normaliza la interfaz usuario-red (UNI), de modo que cada proveedor implementa su propio mecanismo de provisión [61]. Esta fragmentación complica la orquestación de QoS extremo-a-extremo si no existe un SDN unificado [57]. De ahí la necesidad de una herramienta ligera capaz de distribuir configuraciones deterministas entre nodos autónomos sin mantener un controlador permanente; diversos trabajos industriales remarcan esta carencia y proponen scripts bajo demanda o *network digital twins* para validar cambios antes de aplicarlos [54], [62].
- **Lógica de configuración autónoma (sin contenedores ni controladores centralizados).** En muchos trabajos se asume un controlador SDN permanente (con stacks

Java/OSGi o contenedores Docker) para administrar la red determinista [27], [58]. Sin embargo, en entornos industriales *constrained* (PLCs, microcontroladores o redes pequeñas) el despliegue de un plano SDN completo resulta inviable por CPU, memoria o simplicidad operativa [58]. Para estos escenarios, la IETF desarrolla **CORECONF**, una interfaz de gestión ligera basada en CoAP/CBOR sobre UDP que ofrece la misma semántica de YANG que NETCONF/RESTCONF, pero con mínimo *overhead* [63]. evaluaron NETCONF, RESTCONF y CORECONF en controladores TSN baratos y evidenciaron que CORECONF reduce tanto el tiempo de configuración como el consumo de energía de los dispositivos [64]. Estos resultados confirman que los protocolos tradicionales (NETCONF sobre SSH/XML) son demasiado pesados en ciertos casos y que se requieren alternativas más ligeras (CoAP/CBOR, UDP) para orquestación distribuida. El configurador e2e propuesto integra esta lógica ligera: se ejecuta puntualmente, calcula rutas y horarios y los entrega directamente a switches/endpoints deterministas mediante CORECONF o ficheros YANG, sin mantener un *daemon* central activo. Así se cubre la brecha de gestión determinista en redes sin infraestructura SDN continua.

- **Latencia *y* ancho de banda como único criterio determinista.** La mayoría de configuradores deterministas satisfacen la *deadline* reservando ancho de banda al peor caso, lo que deja capacidad ociosa [65]. Se ha medido que ventanas TT holgadas desperdician hasta un 30 % de enlace disponible [66]. Por ello, el borrador de requisitos de escalado de DetNet exige que el algoritmo de ruta considere, de forma conjunta, retardo máximo y capacidad residual [67]. El configurador *e2e* cubre esta brecha, calculando rutas que cumplan la *deadline* sin sobre aprovisionar banda.

En conclusión, las soluciones actuales han sentado bases importantes (SDN para TSN, algoritmos de ruteo QoS, integración OPC UA, etc.), pero adolecen de complejidad o rigidez que dificultan su adopción en ciertos escenarios. El nuevo configurador extremo a extremo buscará automatizar la configuración determinista sin necesidad de un controlador pesado, implementando lógica propia para seleccionar rutas y programar recursos con criterios integrados de latencia y ancho de banda, llenando así las brechas identificadas en la gestión de redes deterministas de próxima generación.

3. ANÁLISIS Y ESPECIFICACIÓN DEL PROBLEMA

3.1. Descripción del servicio a configurar

En el plano de control de PREDICT-6G, el módulo *E2E Configurator* recibe la descripción de cada nuevo servicio (formato JSON) desde el *Service Automation* Management Service del AICP (AI-driven Inter-domain Control Plane) [24]. A partir de esa petición, el configurador calcula la ruta y los recursos deterministas necesarios y los distribuye al resto de Management Services implicados (p. ej., *Path Computation*, *Resource Configuration*) para que éstos apliquen la configuración en sus dominios tecnológicos. De esta forma, la petición JSON constituye la entrada principal del configurador y habilita la provisión automática de servicios *end-to-end* con garantías de latencia, fiabilidad y ancho de banda. La estructura del JSON recibido por el configurador, a través de una petición HTTP POST, permite describir las necesidades del servicio, incluyendo tanto su identificación como sus parámetros de calidad de servicio (QoS) y características de tráfico. En base a esta información, el sistema es capaz de calcular una ruta viable en la red y, como objetivo final, aplicar la configuración adecuada en los nodos implicados, automatizando la comunicación con los routers para garantizar que el servicio pueda desplegarse correctamente.

A continuación se describen las principales secciones del mensaje JSON:

3.1.1. Identificación y Control del Servicio

- **“e2e_service_id”:** Es el identificador único del servicio extremo a extremo. Este identificador permite rastrear y gestionar cada servicio de forma individual en todo el ciclo de vida de la configuración.
- **“discard”:** Este campo actúa como una señal de control para determinar si el servicio debe ser activado o descartado. Un valor de 0 indica que se desea la configuración del servicio, mientras que un valor 1 activa un procedimiento de eliminación, liberando los recursos asignados previamente en la red.

3.1.2. Características de Calidad de Servicio (QoS)

- **“qos_characteristics”:** En esta sección se especifican los parámetros que definen el nivel de calidad que debe proporcionar la red para el servicio solicitado. Algunos de los más relevantes para este trabajo son:
  - **“td_reliability”:** Indica el nivel de fiabilidad requerido. En este caso, si el

valor es mayor o igual a 98, se asume que el tráfico es de tipo TSN (Time-Sensitive Networking).

- **“td_packet_loss”:** Porcentaje máximo tolerable de pérdida de paquetes.
- **“td_delay”:** Especifica la latencia máxima permitida para el servicio. Este parámetro es crítico en redes DetNet, ya que garantiza la sincronización.
- *Otros parámetros:* “priority”, “td_rtt”, “td_jitter”, “burst_arrival_time_window”, “burst_completion_time_window”.

3.1.3. Características del Tráfico

- **“traffic_characteristics”:** Esta sección del JSON describe las propiedades del tráfico de datos que se va a transmitir:
  - **“maximum_flow_bitrate”:** Este valor especifica el ancho de banda requerido para el servicio.
- *Otros parámetros:* “direction”, “periodicity”, “burst_size”.

En resumen, el servicio a configurar (recibido en formato JSON es la pieza central en el diseño del configurador de recursos E2E. A partir de esta entrada, el sistema analiza los requisitos, selecciona una ruta viable y aplica la configuración necesaria en los routers implicados para garantizar su cumplimiento. Esta capacidad de traducir automáticamente las necesidades del servicio en configuraciones de red reales es precisamente lo que valida la utilidad y la eficacia del configurador propuesto en este trabajo.

3.2. Requisitos del sistema

El diseño e implementación del configurador de recursos E2E para redes deterministas basadas en DetNet debe cumplir con un conjunto de requisitos que garantizan su correcto funcionamiento y rendimiento dentro del entorno de red. Estos requisitos se dividen en dos categorías: funcionales y no funcionales.

3.2.1. Requisitos Funcionales

Los requisitos funcionales describen las capacidades y funcionalidades que debe proporcionar el sistema para cumplir su propósito:

- **RF1 - Recepción de peticiones:** El sistema debe ser capaz de recibir solicitudes de configuración de servicio a través de una API REST, en formato JSON estructurado.

- **RF2 - Análisis de la solicitud y selección de ruta:** Una vez recibida la petición, el sistema debe interpretar correctamente los identificadores, características de tráfico y requisitos de calidad de servicio (QoS), y ejecutar un algoritmo de selección de ruta que cumpla con los parámetros establecidos (como latencia máxima y ancho de banda mínimo).
- **RF3 - Verificación de disponibilidad de recursos:** Antes de asignar un camino, el sistema debe comprobar que los enlaces seleccionados cuentan con los recursos suficientes (ancho de banda disponible, por ejemplo) para garantizar la viabilidad del servicio.
- **RF4 - Configuración de los nodos de la red:** El configurador debe aplicar los cambios necesarios en los routers o nodos implicados, generando y transfiriendo los ficheros de configuración adecuados para establecer correctamente el servicio E2E.
- **RF5 - Liberación de recursos:** Si un servicio es descartado o finaliza, el sistema debe liberar automáticamente los recursos reservados en la red y actualizar la configuración de los nodos afectados.
- **RF6 - Gestión de errores:** El sistema debe detectar posibles errores durante la recepción, procesamiento o configuración (por ejemplo, JSON mal formateados o falta de recursos)
- **RF7 - Persistencia y seguimiento del estado:** El sistema debe mantener un registro actualizado de todos los servicios activos, permitiendo su trazabilidad, posible eliminación y actualización, almacenando la información relevante en un fichero estructurado.

3.2.2. Requisitos No Funcionales

Los requisitos no funcionales definen las características de calidad que debe cumplir el sistema para garantizar su eficiencia y fiabilidad:

- **RNF1 - Disponibilidad:** El configurador de recursos debe estar disponible en al menos un 99.9 % del tiempo para garantizar la continuidad del servicio en entornos críticos.
- **RNF2 - Escalabilidad:** El sistema debe ser capaz de manejar múltiples solicitudes simultáneas sin degradar su rendimiento.
- **RNF3 - Compatibilidad:** El sistema debe ser compatible con diferentes arquitecturas de red y dispositivos de red.

- **RNF4 - Robustez:** El configurador debe ser capaz de manejar situaciones de fallo sin comprometer el servicio, implementando mecanismos de recuperación y redundancia.
- **RNF5 - Seguridad en la configuración remota:** Dado que el sistema interactúa directamente con los nodos de red y transfiere archivos de configuración, es esencial garantizar mecanismos de autenticación, integridad de datos y protección frente a accesos no autorizados.

Estos requisitos garantizan que el sistema cumpla con su propósito de gestionar recursos de forma eficiente en una red determinista, asegurando calidad de servicio y cumplimiento de los parámetros establecidos por los servicios E2E.

3.3. Desafíos teóricos

El desarrollo del configurador *e2e* implica superar varios retos conceptuales:

- **Comprender la arquitectura DetNet y sus garantías de servicio**. Requiere asimilar los bloques funcionales y los conceptos de PREOF y flujos deterministas (véase Sección 2.1.3, pág. 9).
- **Relacionar esa arquitectura con la de PREDICT-6G**. El *E2E Configurator* debe integrarse con los servicios del plano de control AICP descritos en la Visión General del sistema (Sección 4.1.1, pág. 26).
- **Modelar la QoS multicriterio (latencia, jitter y ancho de banda)**. Es necesario mapear los campos JSON a restricciones formales, como se detalla en la Caracterización de QoS (Sección 3.1.2, pág. 19).
- **Resolver el ruteo bajo restricciones deterministas**. Conlleva analizar la complejidad de algoritmos y heurísticas presentados en el Diseño del Algoritmo de Selección de Ruta (Sección 4.3, pág. 33).
- **Garantizar coherencia en la configuración distribuida sin SDN permanente**. Implica coordinar reservas de recursos y actualizaciones en los nodos (véase Gestión y Control de Recursos, Sección 3.4.3, pág. 23).

3.4. Desafíos Técnicos del desarrollo

El desarrollo e implementación del configurador de recursos E2E para redes deterministas basadas en DetNet supone enfrentarse a una serie de retos técnicos relevantes. A continuación, se presentan los principales retos a superar en este proyecto:

3.4.1. Procesamiento Eficiente de Peticiones JSON

Uno de los primeros retos abordados fue el diseño de un sistema capaz de recibir peticiones en formato JSON de forma fiable. Esto implica no solo aceptar y almacenar la información entrante, sino también validar su estructura y contenido antes de iniciar cualquier tipo de procesamiento.

Además, se ha contemplado la posibilidad de recibir archivos mal formateados o incompletos. Por ello, el sistema incorpora mecanismos de control de errores que evitan fallos en cadena y permiten gestionar de forma controlada situaciones inesperadas.

3.4.2. Selección de rutas bajo restricciones de QoS

Seleccionar el camino óptimo para un servicio E2E no es fácil, especialmente cuando existen restricciones en cuanto a latencia o ancho de banda. El algoritmo desarrollado no solo debe encontrar rutas viables en la red, sino también asegurarse de que cada enlace del trayecto cumple con los requisitos del servicio solicitado.

3.4.3. Gestión y control de recursos en la red

Para garantizar que la red pueda sostener los servicios configurados sin sobrecarga, es fundamental desarrollar un mecanismo eficiente para verificar la disponibilidad de recursos en cada nodo y enlace. Esto implica:

- Asegurar la correcta reserva y liberación de recursos.
- Prevenir la sobreasignación de ancho de banda en los enlaces.

Pero la gestión no termina en la reserva: también hay que liberar correctamente los recursos cuando un servicio finaliza o se descarta. De lo contrario, se corre el riesgo de congestionar artificialmente la red.

3.4.4. Interacción con los Routers y Configuración de la Red

El configurador debe comunicar y aplicar automáticamente las configuraciones en los routers de la red. En la plataforma se emplean **routers DetNet de código abierto basados en Linux eBPF/XDP**, capaces de ejecutar funciones TSN como Frame Replication and Elimination for Reliability (FRER) y PREOF en el propio kernel, con baja latencia [68]. Estos routers permiten inyectar reglas de DetNet sin requerir hardware propietario.

Para garantizar la consistencia se exige que:

- Las configuraciones aplicadas en los routers sean coherentes con la ruta óptima seleccionada.

- Se eviten inconsistencias en caso de fallo de comunicación con los dispositivos.

3.4.5. Rendimiento, concurrencia y escalabilidad

Dado que el sistema está pensado para operar en escenarios críticos, debe ser capaz de gestionar múltiples solicitudes simultáneamente. Para ello, se ha optado por una arquitectura multihilo, donde distintos componentes (recepción HTTP, monitorización de directorios, vigilancia de servicios activos) operan en paralelo sin bloquearse entre sí.

Esta capacidad de concurrencia es clave para garantizar que el configurador pueda escalar en función de la carga del sistema, sin deteriorar su rendimiento ni comprometer la calidad de los servicios ya activos. Además, se ha cuidado el uso eficiente de recursos como CPU y memoria, evitando procesos innecesarios que puedan convertirse en cuellos de botella.

3.4.6. Seguridad en la configuración

El sistema se conecta directamente a los nodos de la red para aplicar configuraciones. Esta interacción remota plantea una serie de retos de seguridad que no pueden pasarse por alto. Por un lado, es necesario autenticar cada conexión, evitando accesos no autorizados. Por otro, resulta vital garantizar la integridad de los archivos transferidos, así como proteger las comunicaciones frente a posibles ataques o interceptaciones.

Aunque en esta fase del proyecto se han utilizado credenciales simples para facilitar la validación, se contempla la incorporación de mecanismos más avanzados de autenticación y cifrado en futuras versiones del sistema.

3.4.7. Supervisión del sistema y análisis del rendimiento

Finalmente, no basta con que el sistema funcione: también debe proporcionar visibilidad sobre cómo lo hace. Contar con métricas que indiquen el tiempo medio de respuesta, el número de servicios activos o el rendimiento de los enlaces permite evaluar el comportamiento del configurador en condiciones reales.

Además, estas métricas ayudan a detectar problemas de forma proactiva y a ajustar los algoritmos o parámetros de red cuando sea necesario. Para facilitar esta tarea, se ha integrado un sistema básico de logging que registra los eventos clave del sistema, y se deja abierta la posibilidad de añadir herramientas gráficas para visualización en tiempo real.

En definitiva, el desarrollo de este configurador ha requerido resolver múltiples desafíos que abarcan desde el tratamiento de datos en la entrada hasta la configuración remota de dispositivos de red. A través de soluciones modulares, eficientes y robustas, se ha con-

seguido construir una herramienta capaz de operar en redes deterministas con garantías, lo cual refuerza la viabilidad del enfoque propuesto en este trabajo.

4. DISEÑO DE LA SOLUCIÓN

4.1. Arquitectura del Sistema

4.1.1. Visión General

El sistema está diseñado para automatizar la configuración de servicios de E2E en redes deterministas (DetNet). El objetivo principal es garantizar que los servicios, definidos mediante una petición JSON, sean procesados y configurados automáticamente sobre la infraestructura de red, asegurando que se cumplan estrictos requisitos de latencia y ancho de banda, cruciales en aplicaciones críticas como vehículos autónomos, automatización industrial y telemedicina.

El sistema está compuesto por diversos módulos que trabajan de manera conjunta, utilizando un enfoque basado en hilos para realizar tareas concurrentes de forma eficiente. A través de este diseño modular, se facilita la escalabilidad y flexibilidad del sistema, permitiendo su adaptación a diferentes entornos de red y necesidades operacionales.

4.1.2. Componentes Principales del Sistema

Servidor HTTP (Flask)

El servidor HTTP implementado en Flask es el punto de entrada para recibir las solicitudes de configuración de servicio. A través del endpoint `/upload-json`, las peticiones JSON son enviadas al servidor, que las valida y almacena en el directorio de entrada `input_files` para su procesamiento posterior. Este servidor es responsable de gestionar las comunicaciones externas con los clientes y de desencadenar el flujo de trabajo.

Directorio de entrada (`input_files`)

El directorio `input_files` es donde se almacenan temporalmente los archivos JSON que se reciben a través de la API.

Monitor de Archivos

El monitor de archivos es un módulo que observa de manera continua el directorio `input_files`. Cuando detecta nuevos archivos, invoca automáticamente el procesamiento de estos archivos, enviándolos al `file_processor.py`. Esto permite que el sistema maneje de manera eficiente los archivos de configuración a medida que llegan.

Procesador de Servicios

Este componente es el encargado de procesar los archivos JSON recibidos. Extrae la información relevante, como la latencia máxima, el ancho de banda requerido, y las características de QoS. Además, ejecuta el algoritmo de selección de ruta para determinar el camino más adecuado para el servicio, basándose en la topología de red y los requisitos especificados. Finalmente, actualiza la base de datos y la representación del grafo de red.

Watchdog

El watchdog monitorea el archivo `database.csv`, que actúa como la base de datos de servicios activos. Cada vez que se detecta un nuevo servicio en la base de datos, el watchdog llama automáticamente al módulo de configuración remota para aplicar la configuración en los routers de la red, asegurando que los servicios se configuren de manera eficiente en la infraestructura.

Base de Datos de Servicios

El archivo `database.csv` almacena los servicios activos y los caminos asignados, junto con otros detalles como el ancho de banda requerido. Este archivo sirve como el núcleo persistente del sistema, permitiendo que el sistema registre y recupere el estado de los servicios configurados.

Topología de Red

La topología de la red está representada en un archivo `graph.gml`, que contiene la estructura de nodos y enlaces de la red, junto con atributos como la latencia y la capacidad de los enlaces. Este grafo es fundamental para el algoritmo de selección de ruta, ya que proporciona la base para calcular las rutas disponibles que cumplen con los requisitos de QoS.

Módulo de Configuración Remota

Este módulo (`configurator.py`) es el encargado de aplicar o desconfigurar los servicios en los routers de la red. A través de SSH/SCP, descarga las configuraciones actuales de los nodos, las modifica según los requisitos del servicio y vuelve a enviarlas a los nodos correspondientes para actualizar la configuración de la red.

Sistema de Logs

El sistema de logs es un componente clave para la trazabilidad del sistema. Todos los eventos y errores generados por el sistema se registran en el archivo `mi_proyecto.log`. Esto no solo facilita la depuración y el análisis de fallos, sino que también proporciona un historial completo de las acciones realizadas en el sistema.

4.2. Lógica del Procesamiento de Peticiones

4.2.1. Recepción de la solicitud y validación

El punto de partida del sistema se encuentra en la recepción de peticiones HTTP por parte del servidor Flask. El sistema expone un endpoint /upload-json que acepta solicitudes POST con contenido en formato JSON. Esta petición representa un nuevo servicio de E2E que se desea configurar en la red.

Cuando el servidor recibe una solicitud, lo primero que hace es comprobar que el cuerpo de la petición contiene datos válidos en formato JSON. En caso contrario, se devuelve un mensaje de error y no se continúa con el proceso.

Si el contenido es válido, el sistema genera automáticamente un nombre de archivo único, que incluye un timestamp, y guarda el contenido JSON en el directorio de entrada `(input_files)`. Esta acción marca el inicio del ciclo de vida del servicio dentro del sistema.

Este primer paso no solo asegura que todas las peticiones queden registradas, sino que también permite desacoplar la recepción de la solicitud de su procesamiento, lo que es clave para garantizar la robustez ante cargas elevadas o tiempos de espera imprevistos.

4.2.2. Flujo de procesamiento y registro del servicio

Una vez almacenado el archivo en el directorio `(input_files)`, entra en acción el módulo de monitorización de archivos, que observa continuamente este directorio. Su misión es sencilla pero esencial: detectar la aparición de nuevos archivos y desencadenar su procesamiento.

Cuando el monitor detecta un archivo nuevo, llama al procesador de archivos, un módulo que se encarga de leer el JSON, interpretar los parámetros del servicio y decidir si se trata de una solicitud nueva o de descarte.

Si se trata de una nueva solicitud, el procesador extrae los valores de calidad de servicio (`(td_delay)`, `(td_reliability)`, etc.) y evalúa si el tráfico requiere tratamiento como TSN. Con esta información, ejecuta un algoritmo de selección de ruta sobre la topología de red definida en `graph.gml` (véase la sección 4.3). Si se encuentra un camino que cumpla los requisitos, se actualiza el grafo (reduciendo el ancho de banda disponible

en los enlaces utilizados) y se registra la nueva entrada en database.csv.

> **Nota.** A lo largo de este trabajo, se entiende por *ruta óptima* aquella que, cumpliendo con los requisitos de ancho de banda en todos sus enlaces, presenta la **mayor latencia total posible dentro del límite establecido**. Este criterio busca preservar las rutas con menor latencia para futuros servicios más exigentes en términos temporales, favoreciendo una distribución más eficiente de los recursos de red.

Si se trata de un descarte (discard = 1), el sistema localiza la entrada correspondiente en la base de datos, libera los recursos previamente reservados en el grafo y elimina la línea de database.csv. Además, activa el proceso de desconfiguración remota para revertir los cambios aplicados en los routers.

En ambos casos, el procesador escribe eventos relevantes en el archivo de log, lo que permite mantener una trazabilidad completa del sistema.

Una vez que el archivo JSON ha sido procesado y la información del servicio queda registrada en la base de datos, entra en juego el **watchdog**, que se encarga de detectar nuevas líneas en database.csv. Cuando identifica un nuevo servicio, activa el módulo de configuración remota, que se conecta a los nodos implicados mediante SCP/SSH, descarga los archivos de configuración actuales, aplica los cambios necesarios y los vuelve a enviar al dispositivo (véase la sección 5.5.2).

Así, la configuración del servicio queda aplicada de forma completamente automática, desde que el cliente envía la solicitud hasta que los routers han sido configurados para dar soporte al tráfico correspondiente.

4.2.3. Diagrama de flujo del sistema

El funcionamiento descrito puede visualizarse a través de una serie de diagramas de flujo que representan con claridad cada paso del proceso, desde la recepción de la solicitud hasta la persistencia del servicio en la red.

A continuación se presentan los diagramas de flujo que ilustran el funcionamiento interno del sistema desde el arranque hasta el procesamiento completo de una petición.

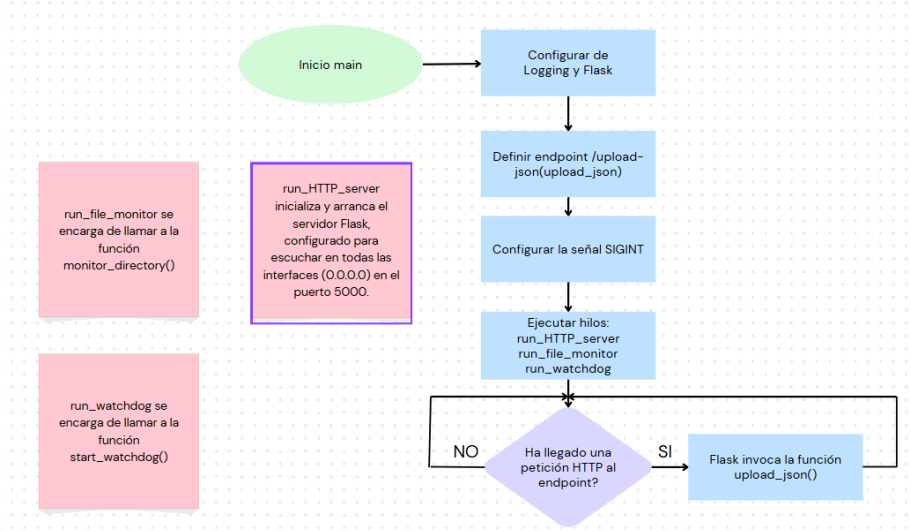


Fig. 4.1. Diagrama de flujo del controlador principal (main.py)

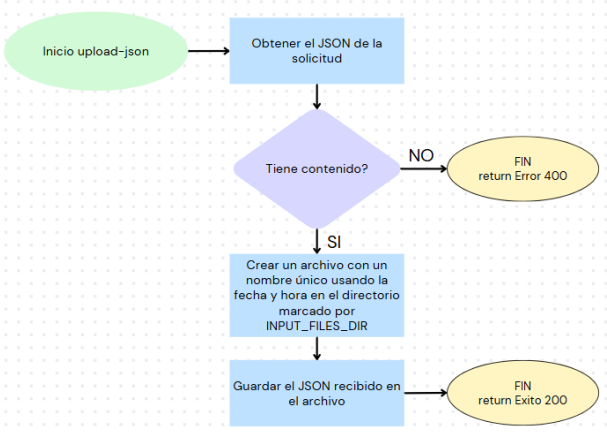


Fig. 4.2. Procesamiento de la petición HTTP en Flask mediante `upload_json()`

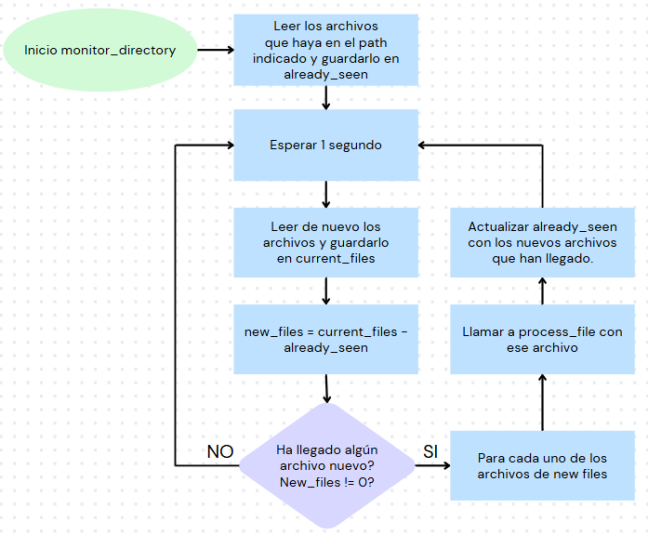


Fig. 4.3. Monitorización del directorio `input_files` y detección de nuevos archivos

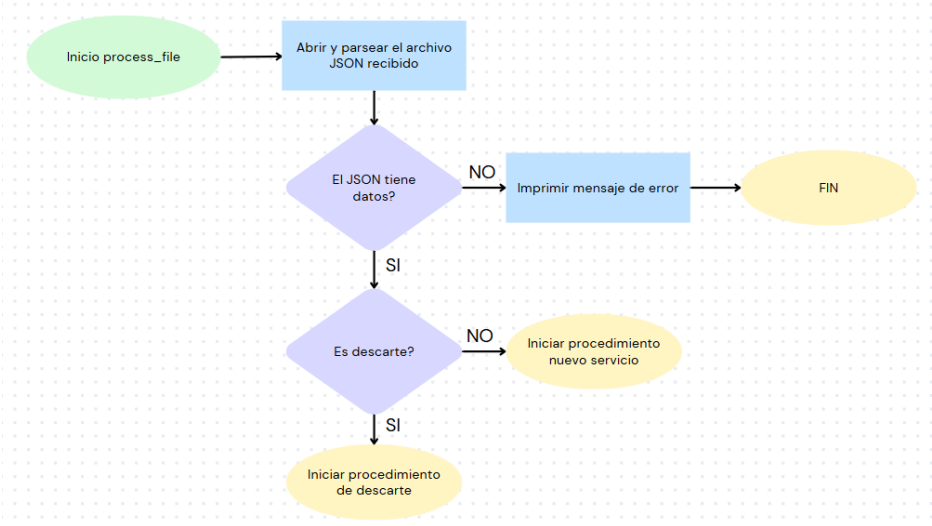


Fig. 4.4. Inicio del procesamiento del archivo JSON detectado

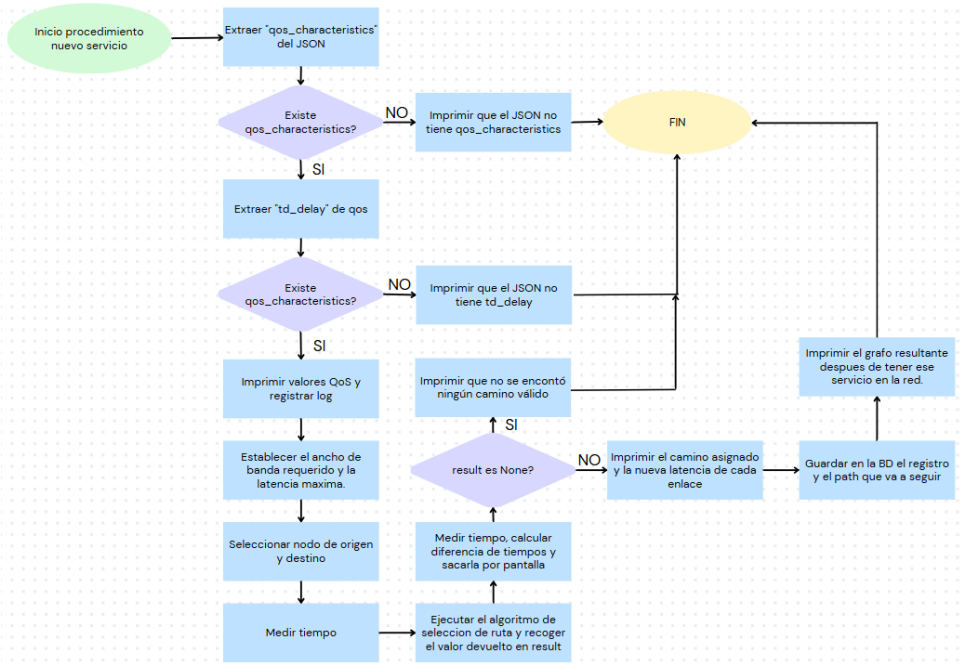


Fig. 4.5. Flujo de procesamiento de un nuevo servicio a partir del JSON recibido

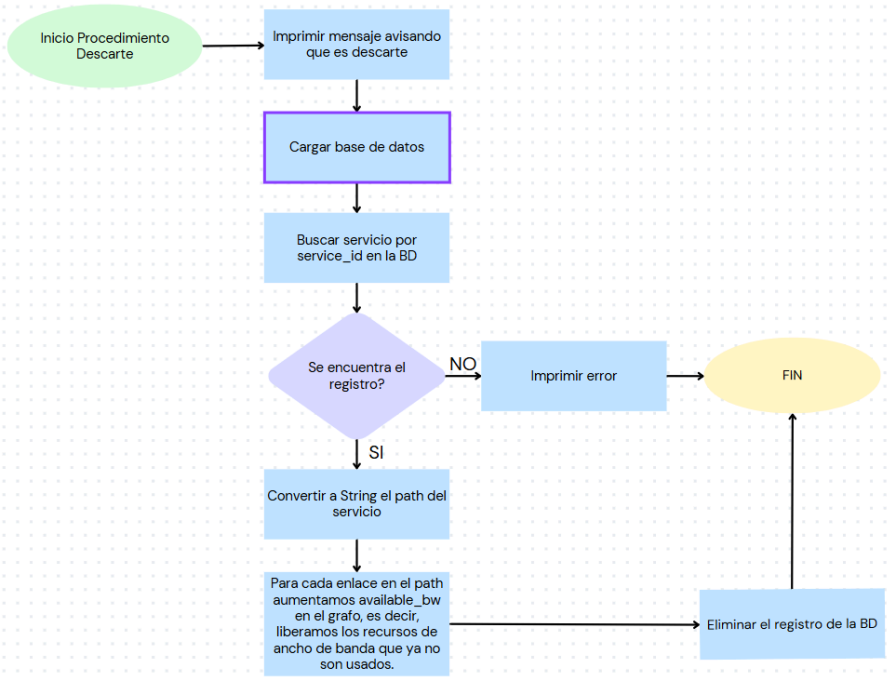


Fig. 4.6. Flujo de eliminación de un servicio ya activo (descartado)

Nota: Los diagramas correspondientes al algoritmo de selección de ruta y a la validación de caminos viables se encuentran en la sección 4.3 dedicada al mecanismo de enrutamiento del sistema. Así mismo los correspondientes a la parte de configuración y deconfiguración de los routers se encuentran en la sección 4.1.2.

Estos diagramas ayudan a entender cómo el sistema está estructurado de forma modular, con responsabilidades bien definidas en cada etapa del flujo. Además, permite identi-

ficar fácilmente puntos clave del sistema, como la validación inicial, la separación entre entrada y procesamiento, o la automatización en la aplicación de configuraciones.

Gracias a este diseño, el sistema es capaz de gestionar múltiples peticiones de manera eficiente, manteniendo una lógica clara y facilitando su mantenimiento o ampliación futura.

4.3. Diseño del Algoritmo de Selección de Ruta

4.3.1. Planteamiento del problema y requisitos

En el contexto de redes deterministas, donde es necesario garantizar la transmisión de flujos con requisitos estrictos de calidad de servicio (QoS), surge la necesidad de un algoritmo que seleccione rutas viables para cada nuevo servicio solicitado. Cada petición de servicio viene acompañada de ciertas características de red, tales como la latencia máxima permitida (`td_delay`) y el ancho de banda requerido (`required_bw`). El algoritmo debe analizar la topología de red actual, representada mediante un grafo en formato GML, y encontrar un camino que cumpla con dichos requisitos.

El reto principal consiste en identificar un camino óptimo dentro de la topología. En este sistema, entendemos como **ruta óptima** aquella que, entre todas las rutas viables que cumplen los requisitos de ancho de banda y latencia, presenta la mayor latencia total posible. Esta elección estratégica busca reservar primero las rutas menos exigentes en cuanto a latencia, dejando disponibles aquellas con menor margen de latencia para servicios que en el futuro puedan requerir condiciones más estrictas.

Este problema implica, por tanto, una doble verificación por cada camino evaluado:

- Que todos los enlaces del camino dispongan de un ancho de banda disponible igual o superior al requerido por el servicio.
- Que la suma de latencias a lo largo del camino no supere el valor máximo de latencia definido para el servicio.

En caso de que existan múltiples rutas viables, el algoritmo seleccionará aquella con la mayor latencia total, aplicando así una política de enrutamiento latency-aware, pero con una visión inversa a la habitual, priorizando rutas "menos óptimas"desde un punto de vista de rendimiento, con el objetivo de optimizar la disponibilidad de la red para futuras solicitudes más exigentes.

4.3.2. Criterio de selección aplicado

La necesidad de encontrar una ruta adecuada para cada servicio en una red determinista requiere un criterio de selección que no solo garantice la viabilidad técnica (es

decir, cumplir los requisitos de latencia y ancho de banda), sino que también sea eficiente aprovechando los recursos de la red.

En este caso se ha adoptado un enfoque personalizado, implementado específicamente para este sistema, con un objetivo concreto: asignar a cada servicio un camino óptimo dentro de las rutas viables, priorizando el ahorro de recursos más críticos.

Este criterio se basa en dos pasos consecutivos:

1. **Filtrado de rutas viables**, descartando todas aquellas en las que:
   - Algún enlace no tenga suficiente ancho de banda disponible.
   - La suma total de latencia supere la latencia máxima aceptable para ese servicio.
2. **Selección del camino óptimo entre los viables**, definido como aquel cuya **suma total de latencias sea la mayor posible**.

   Nota. A lo largo de este trabajo, se entiende por *ruta óptima* aquella que, cumpliendo con los requisitos de ancho de banda en todos sus enlaces, presenta la **mayor latencia total posible dentro del límite establecido**. Este criterio busca preservar las rutas con menor latencia para futuros servicios más exigentes en términos temporales, favoreciendo una distribución más eficiente de los recursos de red.

El algoritmo implementado sigue esta filosofía de diseño: recorre todos los caminos simples entre origen y destino, evalúa su viabilidad mediante la función `validate_path()`, y selecciona la ruta viable con mayor latencia acumulada. Este enfoque permite tomar decisiones eficaces mediante una lógica comprensible y controlada, como se refleja en los diagramas de flujo de las funciones `route_selection()` y `validate_path()` (véase la sección 4.3.3).

Este diseño ha demostrado ofrecer un buen equilibrio entre simplicidad, control lógico y adecuación al contexto, ya que permite tomar decisiones conscientes sobre el uso de recursos críticos. Además, al haber sido diseñado e implementado de forma personalizada, su integración con el resto del sistema resulta natural y su mantenimiento, más manejable.

4.3.3. Diagrama de flujo del algoritmo

A continuación, se presentan los diagramas de flujo que representan el funcionamiento interno del algoritmo de selección de ruta implementado en este sistema. Este algoritmo combina una búsqueda de caminos viables con un criterio de selección que prioriza la mayor latencia posible dentro de los requisitos establecidos.

El primer diagrama muestra la lógica principal del algoritmo, que explora todas las rutas posibles entre el nodo origen y el nodo destino. Por cada camino detectado, se llama a un subprocedimiento que determina si cumple los requisitos necesarios.

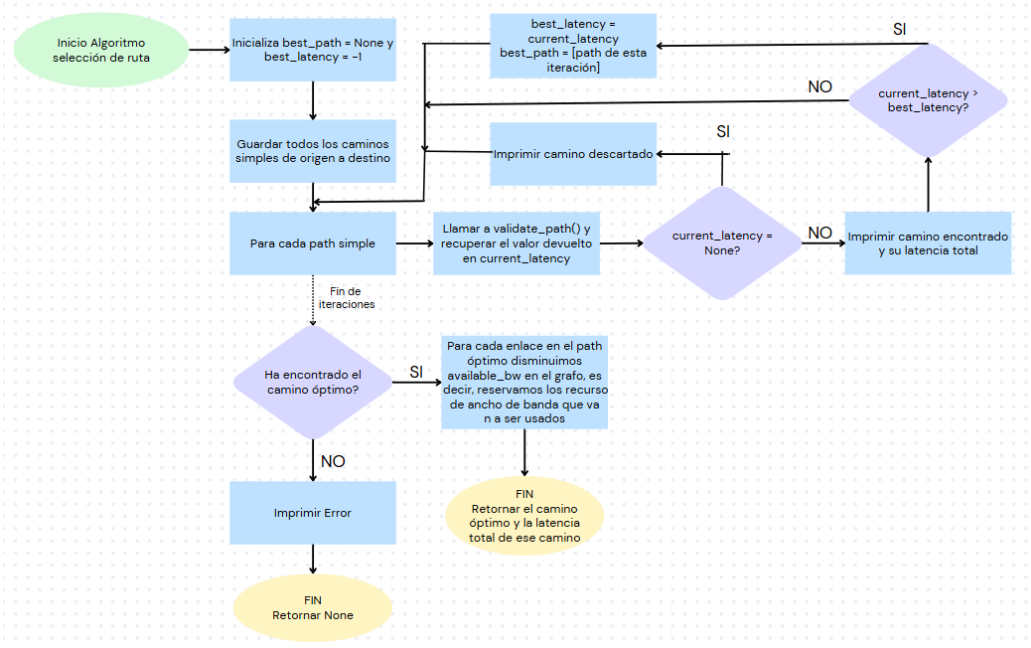


Fig. 4.7. Diagrama de flujo del algoritmo principal de selección de ruta.

La siguiente figura detalla el procedimiento de validación de cada camino, comprobando si el ancho de banda disponible en todos los enlaces es suficiente y si la latencia total del recorrido no supera el límite permitido.

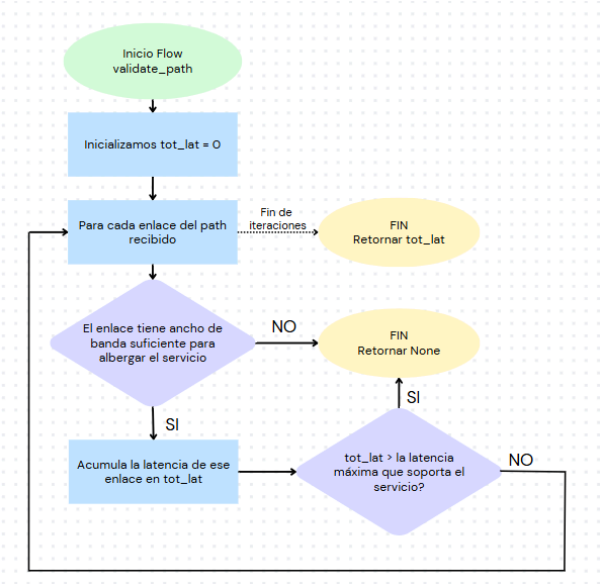


Fig. 4.8. Diagrama de flujo del subprocedimiento de validación de caminos (‘validate_path‘).

4.3.4. Alternativas evaluadas y justificación

Durante el diseño del algoritmo de selección de ruta, se exploraron distintas alternativas que podrían haber sido implementadas en lugar del enfoque actual. A continuación,

se detallan brevemente las opciones consideradas y los motivos por los que fueron finalmente descartadas:

1. **Algoritmo de etiquetado MCP (Minimum Cost Path):** basado en explorar rutas de coste mínimo, se descartó por no considerar directamente restricciones de tipo QoS como la latencia máxima o la disponibilidad de ancho de banda. Su integración con los requisitos estrictos del sistema requería adaptaciones no triviales.

2. **Algoritmo de k-caminos con filtrado:** esta alternativa consistía en calcular un conjunto limitado de caminos posibles (por ejemplo, los 5 caminos más cortos) y aplicar posteriormente un filtrado por latencia y ancho de banda. Aunque teóricamente factible, se consideró más complejo de controlar, y su rendimiento dependería excesivamente de la selección previa de los caminos, lo que podría derivar en un mal aprovechamiento de rutas menos convencionales pero igualmente viables.

3. **Algoritmo personalizado basado en pesos por enlace:** esta propuesta consistía en definir un peso para cada enlace utilizando la fórmula:

$$w_{AB} = \frac{BW_{\text{enlace}}}{\text{latencia_máxima_requerida} - \text{latencia}_{AB}}$$

   La lógica del algoritmo era que, en cada nodo, se seleccionase el siguiente salto en función del mayor peso disponible entre los enlaces adyacentes, continuando así hasta alcanzar el destino.

   Sin embargo, esta estrategia presentaba fallos evidentes cuando se evaluaban rutas completas. A modo de ejemplo, si se define una latencia máxima permitida de 4 y todos los enlaces tienen un ancho de banda de 100, la fórmula produce resultados incorrectos. Consideremos esta red de prueba:

   - $w_{AB} = \frac{100}{4-2} = 50$
   - $w_{AD} = \frac{100}{4-3} = 100$
   - $w_{DC} = \frac{100}{4-3} = 100$

   En este caso, el algoritmo seleccionaría el camino $A \rightarrow D \rightarrow C$, cuya latencia total es $3 + 3 = 6$, superando así el límite permitido de 4. Aunque los pesos locales parecen prometedores, la suma global de latencias invalida el camino, lo que demuestra que este enfoque no es apto para nuestro caso.

Ante estas limitaciones, se optó por un algoritmo propio diseñado específicamente para este sistema.

Además, al haber sido desarrollado específicamente para este proyecto, este algoritmo permite un mayor control y comprensión de su funcionamiento interno, lo que facilita futuras modificaciones, depuración o incorporación de mejoras adaptadas a escenarios específicos.

4.3.5. Propuesta de mejora y optimización

Si bien el algoritmo implementado cumple correctamente con los requisitos establecidos y se comporta de forma coherente ante diferentes escenarios, existen varios aspectos susceptibles de mejora, tanto a nivel de rendimiento como de eficiencia global del sistema.

Tiempo de ejecución. Cuando el número de nodos crece, el algoritmo evalúa todos los caminos simples entre origen y destino, lo que conlleva un coste elevado en topologías densas o con muchos servicios concurrentes. Para mitigar este efecto, se plantea:

- limitar el número de caminos explorados mediante técnicas de poda temprana;
- utilizar estructuras de datos más eficientes en el seguimiento de caminos viables;
- aplicar heurísticas que prioricen rutas con mayor probabilidad de éxito, evitando evaluaciones redundantes.

Adaptabilidad a perfiles de tráfico. En su estado actual, todos los servicios se tratan del mismo modo, lo que puede resultar subóptimo en entornos con flujos heterogéneos (críticos y no críticos). Se propone incorporar un sistema de priorización que ajuste dinámicamente el criterio de selección según las características del servicio (deadline, tolerancia a pérdidas, etc.). En esta línea, la IETF ha descrito algoritmos de *Traffic Engineering* para DetNet que consideran explícitamente las colas y el mecanismo CQF de RFC 9320 [21]. Dichos algoritmos calculan rutas de capa 3 con garantías de latencia, pero no configuran las puertas TAS ni los perfiles de cola TSN de los dominios de capa 2. Gracias al diseño modular del *E2E Configurator*, el algoritmo de ruta está encapsulado y puede sustituirse por estas versiones avanzadas, o por futuras basadas en IA, sin impactar en el resto de componentes (procesamiento de peticiones, configuración remota o *logging*).

Inteligencia artificial y aprendizaje automático. Mediante el análisis histórico de rutas y requisitos, el sistema podría aprender patrones de selección óptima y predecir rutas viables sin explorar exhaustivamente el grafo. Este enfoque reduciría aún más el tiempo de cálculo y permitiría adaptarse en tiempo real a la evolución de la carga de red.

En conjunto, estas mejoras refuerzan la escalabilidad y flexibilidad del sistema, facilitando su aplicación en escenarios más complejos o exigentes.

4.4. Diseño de la configuración y deconfiguración

Una vez determinada la ruta óptima para un servicio E2E y el watchog ha detectado una nueva entrada en la base de datos de servicios (véase sec- ción 5.5.1), el sistema debe aplicar reglas específicas en cada uno de los nodos del camino, de forma que el tráfico asociado al servicio pueda transitar por la red. Esta lógica de configuración se basa

en modificar dos archivos JSON en cada nodo: `DetNetData.json` y `newFlows.json`. Estos archivos contienen las reglas que dictan cómo debe tratarse el tráfico.

El sistema clasifica a cada nodo según su posición dentro del camino y le asigna una acción. Es importante entender el rol que desempeña cada nodo dentro de una ruta DetNet. Para ilustrar esto, se introduce a continuación una imagen simplificada del escenario típico:

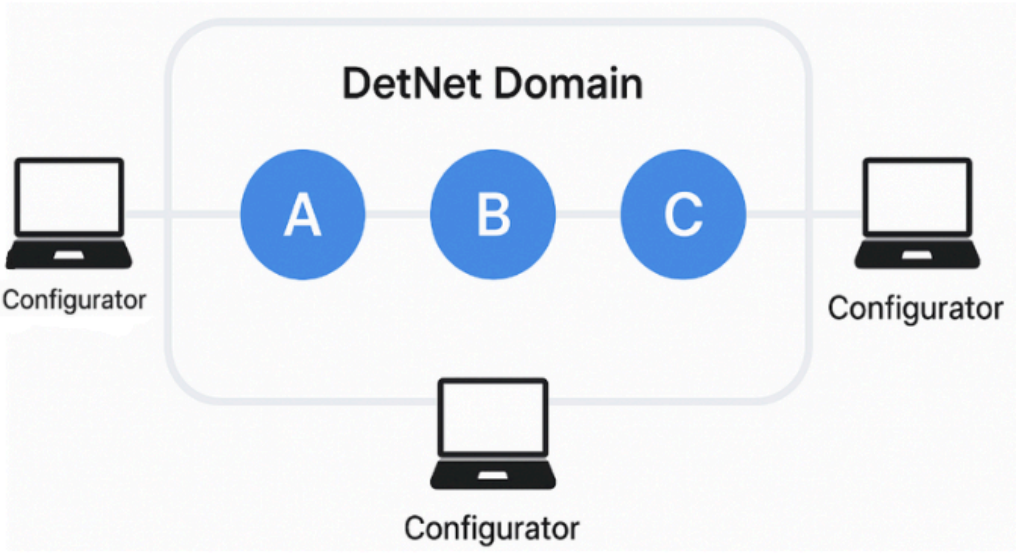


Fig. 4.9. Esquema conceptual de una ruta DetNet con tres nodos intermedios (A, B, C) entre configuradores.

En este ejemplo, un servicio de E2E transita a través de tres nodos de red: A, B y C. Supondremos que el camino óptimo para un servicio entrante está compuesto por estos tres nodos, en el siguiente orden:

$$A \rightarrow B \rightarrow C$$

El nodo A representa el primer nodo del camino (ingreso al dominio DetNet), el nodo B actúa como nodo intermedio, y el nodo C es el último nodo del dominio antes de llegar al destino final (ya sea otro configurador o un equipo final).

> **Nota.** Aunque en la imagen se representan dos configuradores distintos, cualquiera de ellos tiene acceso SSH a todos los nodos de la red. Lo importante es el orden del camino asignado: el configurador se conectará individualmente a cada nodo de dicho camino, sin importar su posición física o lógica.

Para cada uno de estos nodos, el sistema debe decidir qué acción debe tomar cuando reciba un paquete que forma parte del nuevo flujo. Estas acciones se definen como:

TABLA 4.1. ACCIONES CONFIGURADAS EN LOS NODOS SEGÚN SU POSICIÓN EN EL CAMINO E2E.

Nodo	Posición en el camino	Acción configurada
A	Primer nodo (inicio del servicio)	`encapsulate`
B	Nodo intermedio	`forward`
C	Último nodo (antes del destino)	`pop`

- **encapsulate**: Se aplica únicamente al primer nodo del camino (en el ejemplo, el nodo A). Su misión es encapsular el tráfico para que se distinga como parte de un flujo DetNet. Esto incluye el etiquetado L2 y L3, así como los puertos fuente y destino.
- **forward**: Aparece en todos los nodos intermedios (como B), cuya función es simplemente reenviar el tráfico hacia el siguiente salto sin modificar los extremos.
- **pop**: Se utiliza en el último nodo de la ruta (en este caso, C). Este nodo debe "desencapsular" el tráfico, es decir, eliminar las etiquetas añadidas para que pueda salir de la red DetNet hacia el destino final.

El proceso de desconfiguración sigue una lógica inversa al de la configuración: una vez que se detecta que un servicio ha sido marcado como `discard = 1`, el sistema elimina todas las entradas asociadas a su `e2e_id` en los archivos de configuración de los nodos que participaron en el camino asignado. Para ello, se consulta la base de datos local (`database.csv`) con el fin de identificar los nodos afectados por ese servicio. Tras la eliminación remota de dichas configuraciones, se procede también a borrar la entrada correspondiente del servicio en la base de datos, asegurando así la coherencia entre el estado lógico del sistema y la configuración real desplegada en la red.

4.5. Componentes Técnicos Adicionales

Además del núcleo del sistema, basado en el procesamiento de servicios y la selección de rutas, el diseño contempla una serie de elementos técnicos que refuerzan su estabilidad, eficiencia y capacidad de automatización. A continuación se describen los principales mecanismos adicionales integrados en el sistema.

4.5.1. Gestión de procesos concurrentes

El sistema está concebido como un conjunto de procesos que operan de forma simultánea, cada uno con una responsabilidad bien delimitada: el servidor HTTP, el monitor de archivos y el watchdog. Esta estructura se implementa a través de hilos de ejecución independientes (threads), gestionados desde el módulo principal o "controlador".

Gracias a este enfoque multihilo, es posible mantener un servidor activo que reciba peticiones, mientras el sistema monitoriza nuevos archivos y procesa configuraciones, todo en paralelo. Esta arquitectura no solo mejora el rendimiento general, sino que permite desacoplar los distintos componentes, facilitando la escalabilidad y la detección de errores en tiempo real.

4.5.2. Manejo de señales y control de interrupciones

Dado que el sistema se ejecuta de forma continua, es fundamental establecer un mecanismo que permita interrumpir su funcionamiento de manera controlada. Para ello, se ha integrado un gestor de señales que detecta eventos como una interrupción manual por teclado (Ctrl+C) y responde finalizando de forma segura todos los hilos activos.

Este control no solo garantiza un cierre ordenado, sino que también previene comportamientos indeseados como procesos zombis o pérdida de recursos no liberados adecuadamente. De esta forma, se preserva la integridad tanto del sistema como de la red que gestiona.

4.5.3. Sistema watchdog y configuración remota

Uno de los elementos clave del diseño es el módulo watchdog, encargado de vigilar de forma continua el archivo que actúa como base de datos de servicios activos (database.csv). Cada vez que se añade una nueva entrada (producto del procesamiento de un JSON válido), el watchdog la detecta y desencadena automáticamente el procedimiento de configuración remota correspondiente.

Este proceso se lleva a cabo de forma autónoma, conectándose a los nodos implicados mediante conexiones SSH/SCP para aplicar los parámetros adecuados. Gracias a esta automatización, se elimina la necesidad de intervención manual en la configuración de red, aumentando la eficiencia y reduciendo posibles errores humanos.

4.5.4. Robustez y tolerancia a errores

Durante el diseño del sistema se ha prestado especial atención a su capacidad de resistir errores comunes sin comprometer su funcionamiento global. Cada componente incluye mecanismos de validación y gestión de excepciones que permiten, por ejemplo:

- Ignorar archivos mal formateados sin detener el procesamiento general.
- Detectar y registrar intentos de configuración de servicios duplicados.
- Manejar rutas no viables devolviendo mensajes de advertencia sin alterar el grafo.

- Capturar errores de conexión o transferencia de archivos y notificar al sistema de logs.

Este enfoque permite al sistema recuperarse de situaciones anómalas y continuar operando, lo cual resulta esencial en entornos donde la continuidad del servicio es prioritaria.

4.5.5. Aspectos de seguridad en la configuración de red

Si bien la funcionalidad de configuración remota resulta potente, también introduce ciertos riesgos de seguridad inherentes a cualquier operación automatizada sobre dispositivos de red. Por ello, el diseño contempla desde el inicio una serie de buenas prácticas destinadas a reducir estos riesgos:

- Las conexiones con los nodos se realizan mediante `SSH`, un protocolo cifrado y seguro.
- El envío de archivos de configuración utiliza `SCP`, que hereda la capa de seguridad de SSH.
- Se centraliza la autenticación a través de credenciales controladas, evitando accesos no autorizados o conexiones desde ubicaciones desconocidas.
- Todo el proceso queda registrado en el sistema de logs, permitiendo una auditoría posterior si fuera necesario.

Aunque el sistema se ha desarrollado en un entorno de pruebas, estas precauciones permiten que, con ajustes mínimos, pueda adaptarse a escenarios reales donde los aspectos de seguridad sean un factor crítico.

Conclusión del capítulo

A lo largo de este capítulo se ha presentado en detalle el diseño de la solución propuesta para la configuración automática de servicios E2E en redes deterministas. Se ha expuesto la arquitectura modular del sistema, describiendo tanto los componentes principales como su interacción interna, lo que permite comprender la lógica de funcionamiento desde una perspectiva de diseño.

Asimismo, se ha profundizado en el algoritmo de selección de ruta, pieza clave del sistema, justificando el criterio aplicado y analizando las alternativas evaluadas. La elección de una estrategia basada en la conservación de rutas críticas —a través de la selección de caminos viables con mayor latencia— refleja una visión orientada a la eficiencia y sostenibilidad del uso de los recursos de red.

Por último, se han detallado una serie de mecanismos técnicos adicionales que refuerzan la robustez, seguridad y autonomía del sistema, como la gestión de hilos, el control de interrupciones, la automatización mediante *watchdog* o la configuración remota de nodos.

En conjunto, este capítulo sienta las bases del funcionamiento del sistema desde una perspectiva de diseño lógico y funcional, preparando el terreno para abordar, en el siguiente capítulo, su implementación concreta en código.

5. IMPLEMENTACIÓN DEL SISTEMA

5.1. Estructura del código y organización del proyecto

La implementación del sistema se ha llevado a cabo utilizando el lenguaje de programación Python, seleccionando esta tecnología por su sintaxis clara, su amplio ecosistema de bibliotecas de red y automatización, y su facilidad para gestionar estructuras de datos como grafos, JSON y CSV.

Tal y como se detalló en el capítulo anterior (véase la sección 4.1), el sistema ha sido concebido con una arquitectura modular basada en componentes concurrentes. Esta filosofía también se refleja en el código fuente, que ha sido organizado siguiendo una estructura de módulos bien definidos, donde cada funcionalidad principal del sistema se encapsula en un archivo independiente. Esta separación no solo facilita el mantenimiento del código, sino que mejora la legibilidad, la escalabilidad y la trazabilidad del sistema.

5.1.1. Distribución de módulos y carpetas

El proyecto se compone de los siguientes elementos principales:

- **`main.py`**: punto de entrada del sistema. Lanza los hilos principales (servidor HTTP, monitor de archivos y watchdog) y gestiona la señal de salida controlada. Corresponde al *controlador principal* descrito en la sección 4.1.1.
- **`file_monitor.py`**: monitoriza el directorio `input_files` en busca de nuevos archivos JSON. Cuando detecta uno nuevo, invoca al procesador. Este módulo implementa la lógica descrita en la sección 4.1.2 como "monitor de archivos".
- **`file_processor.py`**: encargado del análisis del JSON recibido, ejecución del algoritmo de selección, validación, registro en la base de datos y gestión de descarte. Implementa el procesador de servicios detallado en la sección 4.1.2.
- **`watchdog_bd.py`**: observa la base de datos en busca de nuevos servicios y desencadena el proceso de configuración remota automática, como se abordó en la sección 4.1.2.
- **`configurator.py`**: módulo que gestiona la generación, descarga, modificación y subida de archivos de configuración a cada nodo de la red. Este componente ejecuta la lógica de configuración remota descrita en 4.1.2.
- **`scp_handler.py`**: gestiona las conexiones SSH/SCP para enviar o recibir archivos desde los routers. Funciona como soporte para el módulo `configurator.py`.

- **`config.py`**: define variables globales como rutas de carpetas, archivos de log y configuraciones generales del entorno de ejecución.
- **`graph.gml`**: archivo que representa la topología de red usada por el algoritmo para encontrar caminos viables. Fue introducido en la sección 4.1.2.
- **`network_info.json`**: contiene información detallada sobre IPs, MACs y vecinos de cada nodo de la red. Es esencial para la configuración punto a punto de los nodos (véase 4.1.2).
- **`database.csv`**: base de datos simple donde se registran los servicios activos y sus parámetros (ruta, ancho de banda, tipo de tráfico, etc.). Actúa como núcleo persistente del sistema, como se explicó en 4.1.2.
- **Directorios de soporte**:
  - `input_files/`: almacena temporalmente los archivos JSON recibidos por el servidor.
  - `logs/`: almacena el archivo `mi_proyecto.log` donde se registra el comportamiento del sistema.

Dependencias utilizadas y entorno de ejecución

El proyecto se implementó en **Python 3.13.0** [69] sobre un entorno local. Las principales bibliotecas fueron:

- **Flask** [70]: servidor HTTP ligero para recibir peticiones `POST`.
- **networkx** [71]: modelado y manipulación de la topología como grafo.
- **paramiko** [72] y **scp** [73]: gestión de sesiones `SSH` y transferencias remotas.
- **matplotlib** [74]: visualización del grafo con latencias y anchos de banda anotados.

5.1.2. Consideraciones generales sobre el desarrollo en Python

Python ha demostrado ser una elección adecuada para este proyecto, no solo por su facilidad de uso, sino por su capacidad para gestionar operaciones de red, archivos estructurados y concurrencia de forma sencilla.

Además, la modularidad nativa del lenguaje ha permitido mantener el proyecto limpio, extensible y fácilmente integrable con otras herramientas si se desea evolucionar el sistema hacia entornos más complejos (como entornos cloud o redes reales de producción).

Este enfoque modular, ya planteado desde la fase de diseño, se ha materializado en una estructura de código clara, trazable y preparada para posibles ampliaciones futuras.

5.2. Implementación del procesamiento de peticiones

Esta sección describe el funcionamiento interno del sistema una vez recibida una solicitud HTTP en forma de archivo JSON, desde su almacenamiento hasta la toma de decisiones de configuración y persistencia en la base de datos. Todos los pasos aquí descritos ya fueron introducidos desde una perspectiva de diseño en el capítulo 4.2, pero en esta sección se profundiza en su implementación concreta en código.

5.2.1. Recepción y almacenamiento del archivo JSON

La función `upload_json()`, implementada en `main.py`, recibe las solicitudes HTTP entrantes. Si la petición contiene un JSON válido, se guarda en el directorio `input_files` con un nombre único generado mediante `datetime.now()`.

```
filename = f"input_{datetime.now().strftime('%Y%m%d%H%M%S')}.json"
filepath = os.path.join(INPUT_FILES_DIR, filename)
with open(filepath, 'w') as json_file:
    json.dump(data, json_file, indent=4)
```

El servidor HTTP corre en un hilo propio, al igual que el monitor y el watchdog, lo que permite que todo el sistema funcione de manera concurrente y no bloqueante.

5.2.2. Procesamiento preliminar y validación de datos

El archivo guardado es detectado por el módulo `file_monitor.py`, que ejecuta `process_file()` sobre cualquier nuevo archivo identificado. En esta función se abre el JSON y se verifica que incluya los campos clave como `e2e_service_id` y `qos_characteristics`.

```
        while True:
        time.sleep(1)  # Espera un segundo entre cada chequeo
        current_files = set(os.listdir(path))

        # Determina archivos nuevos
        new_files = current_files - already_seen
        for file_name in new_files:
            file_path = os.path.join(path, file_name)
            logging.info(f"Archivo detectado, procesando archivo...:
                {file_path}")
            process_function(file_path)  # Procesa el nuevo archivo

        already_seen = current_files  # Actualiza la lista de
            archivos vistos para que sea continuo
```

5.2.3. Carga de topología y sincronización con base de datos

Antes de procesar cualquier archivo de entrada, el sistema inicializa la estructura de red leyendo el archivo `graph.gml`, que contiene la topología de nodos y enlaces. Esta operación se realiza de forma centralizada al cargar el módulo `file_processor.py`, generando un grafo de tipo `Graph()` que representa la red.

```
GLOBAL_GRAPH = read_graph()
GLOBAL_GRAPH = update_graph_from_db(GLOBAL_GRAPH)
```

La función `read_graph()` parsea el archivo y normaliza los nodos como enteros, garantizando la coherencia del identificador. Posteriormente, la función `update_graph_from_db()` recorre las entradas en `database.csv` (si existe), y ajusta el ancho de banda disponible en los enlaces según los servicios ya registrados.

Este paso es clave para garantizar la persistencia: si el sistema se reinicia, la red se reconstruye con un estado consistente, teniendo en cuenta los recursos reservados por servicios anteriores.

5.2.4. Lectura del archivo y validación del servicio

Una vez el grafo está en memoria, se inicia el procesamiento del archivo recibido. La función `process_file()` comienza leyendo y parseando el JSON desde el archivo detectado por el monitor:

```
with open(file_path, 'r') as f:
    data = json.load(f)

if not data:
    print(f"El archivo {file_path} esta vacio o no contiene datos.")
    return
```

Se extraen los parámetros clave: el identificador del servicio (`e2e_service_id`) y la bandera de descarte (`discard`). Antes de decidir si se trata de una alta o eliminación, el sistema comprueba si el servicio ya está registrado en la base de datos:

```
service_id = data.get("e2e_service_id")
discard_flag = data.get("discard", 0)

permitido, mensaje = check_service_status(service_id, discard_flag)
if not permitido:
    print(mensaje)
    logging.error(mensaje)
    return
```

La función `check_service_status()` evalúa la validez de la operación según el contexto actual: si el servicio ya existía o si se intenta descartar un servicio inexistente. Esto actúa como un primer control de integridad lógica.

Si la operación es válida, el flujo se bifurca: si `discard = 1`, se pasa al procedimiento de liberación de recursos. En caso contrario, se continuará con el análisis del servicio y la ejecución del algoritmo de selección de ruta (Véase en detalle en la siguiente sección).

5.2.5. Determinación del tipo de operación: alta o descarte

Una vez verificada la validez del servicio mediante `check_service_status()`, el sistema decide qué procedimiento aplicar en función del campo `discard` presente en el archivo JSON.

- Si el valor de `discard` (guardado en la variable `discard_flag`) es igual a `1`, se considera una solicitud de eliminación. El sistema buscará el servicio en la base de datos, liberará los recursos reservados en el grafo y procederá con su desconfiguración remota (ver sección 5.3).
- En cualquier otro caso (por defecto, `0`), se trata de una solicitud de alta. El flujo continúa hacia el análisis de las características QoS y la ejecución del algoritmo de selección de ruta (ver sección 5.4).

5.3. Procedimiento de descarte y liberación de recursos

Cuando una petición JSON incluye el campo `discard = 1`, el sistema interpreta que se trata de una solicitud de eliminación de un servicio previamente configurado. Esta operación implica no solo eliminar el registro del servicio en la base de datos, sino también liberar los recursos reservados en la red y revertir la configuración aplicada en los nodos involucrados.

5.3.1. Liberación de recursos en la topología de red

El primer paso consiste en localizar la entrada correspondiente al servicio en `database.csv`, a partir de su `e2e_service_id`. Una vez identificado, se extrae el camino que había sido asignado al servicio y el ancho de banda reservado. A continuación, se recorre cada enlace del camino y se incrementa la disponibilidad de ancho de banda, liberando así los recursos ocupados por el servicio descartado.

```
for i in range(len(path_nodes)-1):
    u, v = path_nodes[i], path_nodes[i+1]
    GLOBAL_GRAPH[u][v]["available_bw"] += stored_required_bw
```

Esta operación asegura que la red vuelva a un estado coherente y disponible para atender futuras solicitudes.

5.3.2. Actualización de la base de datos

Después de liberar los recursos en el grafo, se elimina la entrada correspondiente del archivo `database.csv`. Esto evita que el sistema intente configurar nuevamente un servicio ya eliminado, y mantiene la base de datos sincronizada con el estado real de la red.

```
rows = [r for r in rows if r[0] != service_id]
with open("database.csv", "w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows(rows)
```

5.3.3. Desconfiguración remota de los nodos implicados

Finalmente, se llama a la función `deconfigure_service()`, que elimina las entradas correspondientes al servicio en los archivos de configuración de cada nodo del camino. Para ello, el sistema se conecta remotamente mediante `SSH`, descarga los archivos de configuración actuales, los modifica localmente y los vuelve a subir, sobrescribiendo la configuración eliminando las partes correspondientes al servicio que quiere ser descartado.

> **Nota.** Cada nodo del camino tiene dos archivos principales: `DetNetData.json` y `newFlows.json`. Ambos deben ser actualizados para garantizar la eliminación total del servicio.

Este proceso garantiza que el sistema no solo libere los recursos en el grafo, sino que también elimine las configuraciones aplicadas en los nodos de red, manteniendo así la coherencia entre la lógica del sistema y el estado real de la infraestructura.

5.4. Ejecución del algoritmo de selección de ruta

Cuando se determina que un servicio recibido no es de descarte (`discard = 0`), el sistema procede a buscar la ruta óptima para alojarlo en la red. Esta decisión desencadena la ejecución del algoritmo de selección de ruta descrito en el capítulo 4.3, el cual ha sido implementado en Python de forma eficiente y modular.

5.4.1. Obtención de parámetros de QoS y datos de red

El archivo JSON recibido contiene los parámetros de calidad de servicio (QoS) y las características del tráfico que permiten determinar si un camino entre origen y destino es viable. Los parámetros más relevantes son:

- `td_delay`: la latencia máxima permitida para el servicio.
- `td_reliability`: si es mayor o igual a 98, el tráfico se considera de tipo TSN.
- `maximum_flow_bitrate`: especifica el ancho de banda requerido, ubicado en el bloque `traffic_characteristics`.

En la implementación, estos valores se extraen directamente del contenido del JSON:

```
td_delay = qos.get("td_delay")
td_reliability = qos.get("td_reliability")
is_\gls{tsn} = td_reliability >= 98

traffic = data.get("traffic_characteristics", {})
required_bw = traffic.get("maximum_flow_bitrate", 0)
```

Además, se identifican los nodos origen y destino a partir del grafo global cargado previamente, que representa la topología de red actual sobre la que se ejecutará el algoritmo de búsqueda.

5.4.2. Evaluación de rutas viables y validación

Una vez extraídos los parámetros necesarios, se ejecuta el procedimiento de búsqueda de rutas entre el nodo origen y el nodo destino. Este procedimiento utiliza la función `all_simple_paths()` de la librería `networkx`, que devuelve todas las rutas posibles sin ciclos entre ambos extremos.

Cada ruta encontrada se valida mediante la función `validate_path()`, cuya misión es comprobar que:

- Todos los enlaces del camino disponen de un ancho de banda disponible mayor o igual al requerido.
- La suma de las latencias de todos los enlaces del camino no supera el valor `td_delay` especificado por el usuario.

El proceso de validación se implementa de forma secuencial, evaluando cada enlace del camino hasta que se detecte una violación de los criterios:

```python
def validate_path(G, path, required_bw, max_latency):
    total_lat = 0
    for i in range(len(path) - 1):
        u, v = path[i], path[i+1]
        if G[u][v]["available_bw"] < required_bw:
            return None
        total_lat += G[u][v]["latency"]
        if total_lat > max_latency:
            return None
    return total_lat
```

Si la ruta pasa ambas comprobaciones, se considera viable y se incorpora al conjunto de posibles rutas candidatas.

5.4.3. Selección de la ruta óptima y actualización del sistema

Tras validar todas las rutas encontradas, se selecciona la que presenta la mayor latencia acumulada dentro del umbral permitido. Esta decisión, tal como se detalló en el capítulo 4.3, responde a una estrategia de conservación de rutas críticas de baja latencia para servicios futuros.

La selección se lleva a cabo durante el mismo bucle de búsqueda, comparando las latencias de las rutas viables y actualizando la mejor encontrada hasta el momento:

```python
for path in nx.all_simple_paths(G,source=source,target=destination):
    current_latency = validate_path(G, path, required_bw,
        max_latency)
    if current_latency is not None and current_latency >
        best_latency:
        best_latency = current_latency
        best_path = path
```

Una vez seleccionada la ruta óptima, se actualiza el grafo global disminuyendo el ancho de banda disponible en cada enlace del camino. También se registra la nueva entrada en la base de datos:

```python
for i in range(len(best_path) - 1):
    u, v = best_path[i], best_path[i+1]
    G[u][v]["available_bw"] -= required_bw
```

```python
with open("database.csv", "a", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow([service_id, " -> ".join(map(str, best_path)),
        required_bw, is_\gls{tsn}])
```

Este procedimiento asegura que la red refleja en todo momento el estado real de los recursos utilizados, permitiendo que el sistema continúe operando de forma coherente y eficiente.

5.5. Automatización mediante watchdog y configuración remota

Uno de los pilares de este trabajo reside en su capacidad para aplicar configuraciones en los nodos de forma autónoma, sin intervención humana. Para ello, se ha implementado un mecanismo de vigilancia continua sobre la base de datos de servicios (`database.csv`), complementado por un módulo de configuración remota que actualiza dinámicamente los archivos de configuración de los nodos implicados.

5.5.1. Monitorización de la base de datos

El fichero `database.csv` actúa como memoria persistente del sistema, almacenando los servicios activos, los caminos que se les han asignado, el ancho de banda por enlace que ocupan en la red y si dicho servicio se considera TSN o no. Para detectar nuevos servicios añadidos, se utiliza un módulo de vigilancia denominado `watchdog`, que escanea continuamente este archivo. Un ejemplo de entrada de esta base de datos sería:

TABLA 5.1. EJEMPLO DE ENTRADA EN LA BASE DE DATOS DATABASE.CSV

ID del Servicio	Camino Asignado	Ancho de Banda (Mbps)	¿Es TSN?
1	$0 \rightarrow 1 \rightarrow 3 \rightarrow 4$	50	True

Nota: Los números en el camino representan los ID de los nodos por los que discurre el servicio.

La implementación de esta funcionalidad se basa en una lectura cíclica del contenido del archivo, identificando diferencias respecto a la última lectura mediante conjuntos (`set()`), lo que permite detectar nuevas líneas agregadas. Este enfoque, aunque sencillo, resulta eficaz para entornos de prueba como el que se contempla en este trabajo:

```
with open(database_path, "r") as f:
    initial_lines = f.readlines()
seen_lines = set(initial_lines)

# En cada iteracion del bucle principal
current_lines = set(f.readlines())
new_lines = current_lines - seen_lines
```

Cuando se detecta una nueva línea, esta se interpreta como un nuevo servicio y se extraen sus parámetros (ID, ruta, ancho de banda y si se trata de tráfico TSN). A continuación, se llama al configurador remoto para aplicar los cambios necesarios en la red.

5.5.2. Envío de configuraciones a los nodos mediante SSH/SCP

La configuración remota de los nodos de la red se realiza mediante conexiones seguras SSH. Este proceso lo gestiona el módulo `configurator.py`, que emplea la biblioteca `paramiko` junto con `scp` para realizar la transferencia de archivos.

El flujo seguido para cada nodo del camino asignado es el siguiente:

1. Establecer conexión SSH con el nodo correspondiente.
2. Descargar los archivos de configuración existentes (`DetNetData.json` y `newFlows.json`).
3. Modificar o añadir la entrada correspondiente al servicio recibido.
4. Volver a enviar los archivos actualizados al nodo remoto, sobrescribiendo los anteriores.

La lógica que determina el contenido exacto de las entradas insertadas en los archivos de configuración —incluyendo qué parámetros se añaden, cómo se construyen y con qué criterios se define su estructura— se analiza con mayor detalle en la siguiente subsección (véase sección 5.5.3).

Toda la comunicación con los nodos se realiza de forma segura y autenticada, evitando accesos no autorizados:

```
client = create_ssh_client(hostname, port, username, password)
with SCPClient(client.get_transport()) as scp:
    scp.get(remote_path, local_path=local_path)
```

5.5.3. Lógica de modificación de archivos de configuración en los nodos

Tal y como se detalla en la sección 4.4, una vez que el sistema ha determinado la ruta que debe recorrer el nuevo servicio, procede a aplicar las reglas de configuración correspondientes en los nodos implicados. Esta tarea la ejecuta la función `configure_service()`, contenida en el módulo `configurator.py`.

La función sigue el siguiente flujo para cada nodo del camino:

1. Establecer una conexión SSH con el nodo utilizando `paramiko`.
2. Descargar los archivos `DetNetData.json` y `newFlows.json` mediante `scp`.
3. Construir una nueva entrada de configuración adaptada al nodo, basándose en su posición dentro del camino y en los datos extraídos del archivo `network_info.json`.
4. Insertar o actualizar la entrada correspondiente al `e2e_id`.

5. Subir los archivos modificados de nuevo al nodo remoto, sobrescribiendo los anteriores.

La lógica que define qué acción corresponde a cada nodo (encapsulate, forward o pop) se basa en su índice dentro de la lista del camino. El código que implementa esta decisión es el siguiente:

```
if index == 0:
    action = "encapsulate"
elif index == len(path) - 1:
    action = "pop"
else:
    action = "forward"
```

Posteriormente, la entrada se construye dinámicamente, incluyendo información de capa 2, y si procede, también de capa 3 y 4. A modo de ejemplo:

```
new_entry = {
    "action": action,
    "maxRep": 2,
    "L2": {
        "src_mac": node_data["mac"],
        "dst_mac": next_hop_mac,
        "flow_id": 12 if \gls{tsn}_flag else 11,
        "pcp": 0,
        "vlan": "true"
    }
}
if action != "pop":
    new_entry["L3"] = {
        "src_ip": node_data["ip"],
        "dst_ip": next_hop_ip
    }
    new_entry["L4"] = {
        "src_port": 4000 if \gls{tsn}_flag else 5000,
        "dst_port": 4000 if \gls{tsn}_flag else 10000
    }
```

Una vez generadas, las configuraciones son reintegradas en los archivos descargados y subidas nuevamente al nodo. Este procedimiento se repite de forma iterativa para cada nodo en el camino.

Gracias a esta implementación, se logra una integración directa entre la lógica del sistema (definida en el capítulo 4.4) y su materialización sobre los dispositivos de red.

5.6. Registro de eventos y trazabilidad del sistema

Uno de los elementos clave en todo sistema automatizado y distribuido es la capacidad de realizar un seguimiento continuo de lo que ocurre durante su ejecución. En este proyecto, se ha implementado un sistema de **logs centralizados** que permite registrar cada acción relevante, facilitando tanto la depuración como la supervisión en tiempo real o posterior.

5.6.1. Sistema de logs

Desde el arranque del sistema, se activa una configuración básica de `logging` en Python que permite escribir entradas en un archivo específico definido en el fichero de configuración `config.py`. Esta configuración se activa mediante la siguiente instrucción:

```
logging.basicConfig(
    filename=LOG_FILE,
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
```

El archivo de log generado (`mi_proyecto.log`) se guarda en la carpeta `logs/`, donde quedan reflejadas las decisiones tomadas por cada módulo del sistema, así como cualquier incidencia ocurrida.

5.6.2. Mensajes relevantes y monitoreo de fallos

A lo largo del código, se insertan múltiples llamadas a:

```
logging.info("Mensaje de logging")
```

Gracias a estos registros, es posible reconstruir la trazabilidad completa de un servicio desde su entrada hasta su configuración en los nodos, identificar errores puntuales en el funcionamiento, y verificar si las acciones automatizadas se han completado correctamente.

Este enfoque hace que el sistema tenga una **mayor transparencia y capacidad de auditoría**, atributos esenciales para una solución que pretende operar de forma autónoma en redes con requisitos críticos.

Conclusión del capítulo

A lo largo de este capítulo se ha descrito con detalle la implementación completa del sistema de configuración automática de servicios E2E en redes deterministas, desarrollada

en Python. Se ha expuesto el flujo que sigue una petición desde su recepción hasta la aplicación de reglas específicas en los nodos de la red, diferenciando claramente entre operaciones de alta y de descarte.

También se ha analizado la ejecución del algoritmo de selección de rutas, los mecanismos de automatización mediante `watchdog` y configuración remota, así como los procedimientos de liberación de recursos y desconfiguración. Por último, se ha detallado el sistema de registro de eventos, fundamental para la trazabilidad, depuración y supervisión de la solución.

Con esta implementación, el sistema adquiere una capacidad operativa real, capaz de gestionar solicitudes de forma autónoma y fiable, cumpliendo con los requisitos de QoS propios de redes críticas como DetNet. Todo ello sienta las bases para su validación experimental, que se abordará en el siguiente capítulo.

6. RESULTADOS Y EVALUACIÓN

6.1. Diseño de los escenarios de prueba

6.1.1. Objetivos de validación y metodología

El propósito principal de este capítulo es evaluar el sistema desde una perspectiva funcional y de rendimiento. Dado que se ha diseñado una arquitectura modular orientada a la automatización de la configuración de servicios E2E en redes deterministas, resulta fundamental comprobar que:

- El sistema es capaz de procesar correctamente peticiones de configuración y descarte.
- El algoritmo de selección de ruta identifica caminos válidos y acordes a los requisitos de QoS.
- El algoritmo de selección de ruta escoge la ruta óptima entre todos los caminos válidos.
- La topología de red se modifica de manera coherente tras cada operación.
- El sistema escala adecuadamente ante un número creciente de nodos y rutas.

Para alcanzar estos objetivos, se han diseñado varios escenarios de prueba, agrupados en dos categorías principales:

1. **Pruebas funcionales**, en las que se verifica el comportamiento general del sistema ante peticiones reales, tanto de alta como de descarte. Estas pruebas incluyen la observación del flujo completo de procesamiento y la validación de los cambios en la base de datos, el grafo y los archivos de configuración remotos.
2. **Pruebas de rendimiento**, centradas en evaluar el comportamiento del algoritmo de selección de rutas bajo condiciones controladas de carga. Para ello, se ha implementado una función que genera topologías aleatorias utilizando el modelo *Erdős–Rényi* [75], lo que permite simular redes de diferente tamaño y densidad, manteniendo parámetros realistas de latencia y ancho de banda.

Ambos conjuntos de pruebas permiten valorar no solo si el sistema cumple su función, sino también hasta qué punto es robusto, escalable y eficiente. Esta dualidad resulta especialmente útil en entornos deterministas, donde tanto la corrección lógica como el tiempo de respuesta son factores clave.

6.1.2. Prueba funcional completa del sistema

Para validar el correcto funcionamiento del sistema en condiciones reales, se ha diseñado una prueba controlada sobre una topología de red fija con cinco nodos, cuyas latencias y capacidades han sido establecidas de forma manual. La Figura 6.1 muestra el estado inicial de la red, donde todos los enlaces tienen una latencia de 2 ms y una capacidad disponible de 100 Mbps.

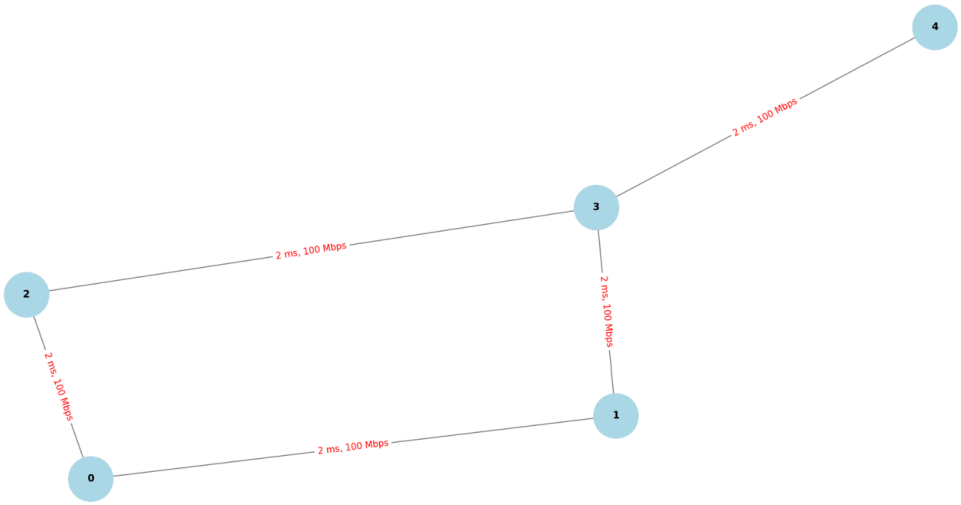


Fig. 6.1. Estado inicial de la red: topología con 5 nodos, enlaces simétricos.

A continuación, se ejecuta un script `tester.py` que simula la llegada de una petición JSON desde un configurador externo. Esta solicitud contiene una latencia máxima asumible de 7 ms y un ancho de banda requerido de 50 Mbps (obtenido del campo `maximum_flow_bitrate`). Como el valor de `td_reliability` es 100, el tráfico se considera TSN.

El sistema detecta la entrada y selecciona uno de los dos caminos viables: $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$ y $0 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Ambos cumplen los requisitos de latencia y capacidad, pero se escoge el primero según la lógica de selección implementada. La Figura 6.2 muestra el nuevo estado del grafo tras esta primera asignación:

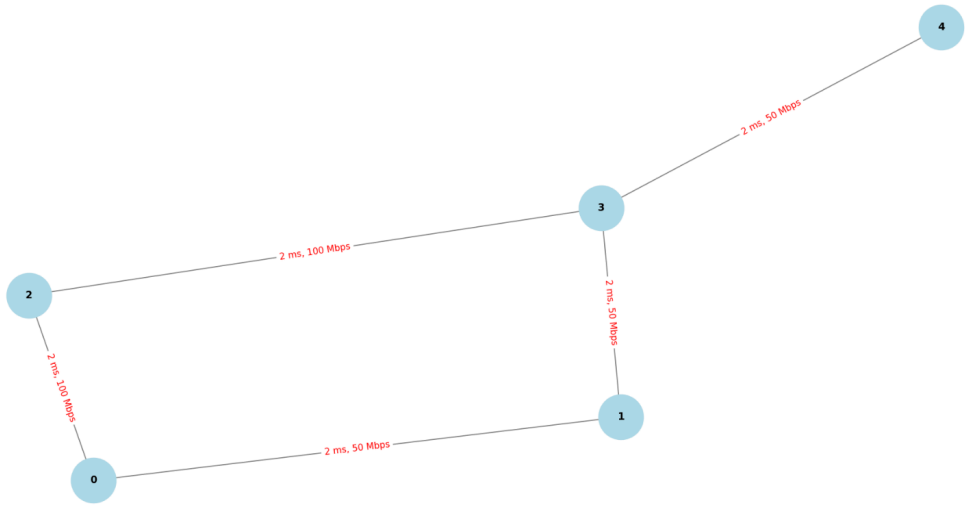


Fig. 6.2. Primer servicio asignado: reducción de capacidad en los enlaces utilizados.

El archivo `database.csv` registra la asignación:

e2e_id	Path	Ancho de banda (Mbps)	TSN
1	$0 \rightarrow 1 \rightarrow 3 \rightarrow 4$	50	True

TABLA 6.1. ESTADO DE LA BASE DE DATOS TRAS EL PRIMER SERVICIO

Si se lanza un segundo servicio idéntico, el sistema asigna nuevamente el mismo camino, ya que aún hay capacidad suficiente. Sin embargo, si se intenta asignar un tercer servicio, ambos caminos serán descartados debido a la insuficiencia de ancho de banda, como se refleja en la terminal con el mensaje:

```
No se encontraron caminos válidos que cumplan los requisitos.
```

La Figura 6.3 muestra la red tras intentar esta tercera asignación:

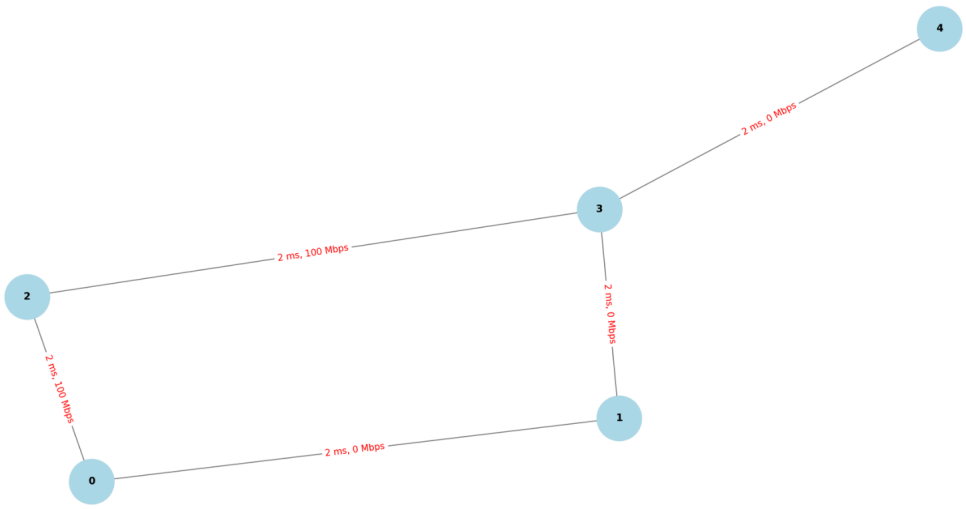


Fig. 6.3. Red sin caminos viables: enlaces sin capacidad suficiente.

En este punto, la base de datos contiene:

e2e_id	Path	Ancho de banda (Mbps)	TSN
1	0 → 1 → 3 → 4	50	True
2	0 → 1 → 3 → 4	50	True

TABLA 6.2. ESTADO DE LA BASE DE DATOS TRAS DOS ASIGNACIONES

Para validar el mecanismo de liberación de recursos, se lanza una solicitud con `discard = 1` para el servicio con ID 2. El sistema localiza la entrada en la base de datos, recupera el camino utilizado, incrementa la disponibilidad en los enlaces correspondientes y elimina la configuración de los nodos implicados. El resultado se observa en la Figura 6.4:

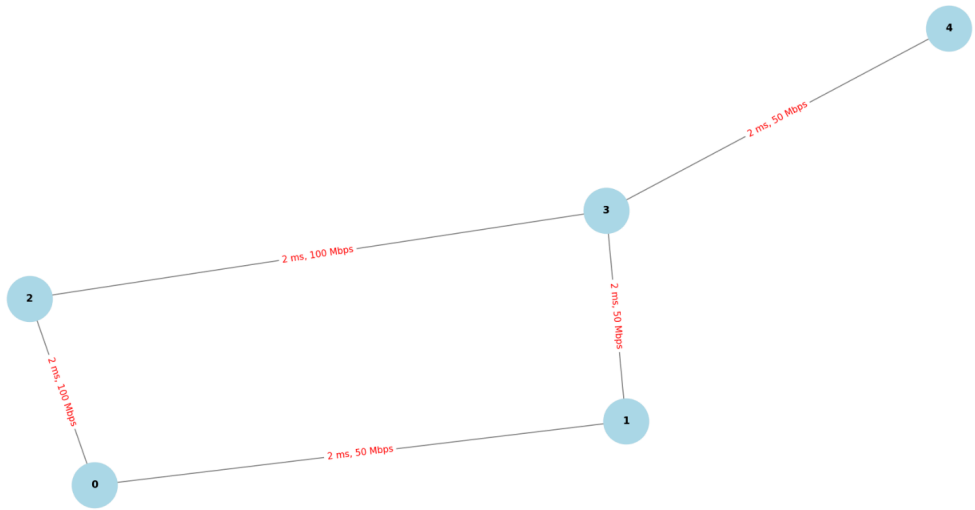


Fig. 6.4. Recuperación de capacidad tras descartar el servicio con ID 2.

La base de datos refleja ahora correctamente el estado actualizado del sistema:

e2e_id	Path	Ancho de banda (Mbps)	TSN
1	$0 \rightarrow 1 \rightarrow 3 \rightarrow 4$	50	True

TABLA 6.3. ESTADO DE LA BASE DE DATOS TRAS DESCARTE

Persistencia ante caídas del sistema

Como prueba adicional, se simula una caída del sistema seguida de su reinicio. Gracias al mecanismo de reconstrucción del grafo desde la base de datos. Al hacer esa prueba el sistema vuelve a cargar el estado anterior sin pérdida de información, manteniendo la disponibilidad correcta en cada enlace.

6.1.3. Verificación de la configuración remota de nodos

6.1.4. Simulación de carga mediante grafos Erdős–Rényi

Tal y como se detalló en la sección 4.3, el algoritmo de selección de ruta implementado tiene como objetivo encontrar el camino óptimo entre origen y destino, cumpliendo unos requisitos mínimos de latencia y ancho de banda. Una vez que se identifican todas las rutas viables, se selecciona aquella cuya suma total de latencia sea máxima, siempre que se mantenga dentro del umbral permitido, lo que denominamos ruta óptima.

Para analizar el rendimiento de este algoritmo desde una perspectiva computacional, es decir, evaluando el tiempo de ejecución en distintos contextos, se ha diseñado un entorno de pruebas específico basado en grafos generados aleatoriamente. Concretamente, se utiliza el modelo de grafos Erdős–Rényi, ampliamente conocido en el ámbito de la teoría de grafos y caracterizado por generar enlaces entre nodos con una probabilidad p configurable. Este enfoque permite obtener redes con estructuras variadas de forma automática y controlada.

La red se genera mediante la función `nx.erdos_renyi_graph(n, p)`, donde n representa el número de nodos y p la probabilidad de que exista un enlace entre cada par de nodos. A cada enlace se le asignan atributos fijos para simular condiciones de red realistas: una latencia aleatoria entre 1 y 10 milisegundos, y una capacidad inicial de 100 Mbps. Además, en caso de que el grafo generado no sea conexo, se extrae automáticamente su componente conexa principal, garantizando así la existencia de al menos un camino entre el nodo origen y el nodo destino.

Este mecanismo proporciona una herramienta versátil para simular escenarios con diferente carga y densidad de enlaces, facilitando el estudio de la escalabilidad y eficiencia del algoritmo sin necesidad de definir manualmente las conexiones en el archivo `graph.gml`.

Nota sobre reproducibilidad. Para garantizar la comparabilidad entre ejecuciones,

el generador aleatorio de grafos ha sido configurado con una semilla fija. Esto asegura que, bajo iguales condiciones de entrada (n y p), se obtenga siempre la misma estructura de red.

Este escenario sintético será la base para las pruebas de carga que se presentan en el siguiente apartado, donde se analizan métricas clave como el tiempo de ejecución del algoritmo en función del número de nodos presentes en la red.

6.2. Análisis de rendimiento del algoritmo

El objetivo de esta sección es evaluar el comportamiento del algoritmo de selección de ruta desde una perspectiva computacional. Específicamente, se analiza el tiempo de ejecución necesario para encontrar la ruta óptima en diferentes contextos de complejidad creciente. Para ello, se emplea la simulación de carga previamente descrita en la sección 6.1.4, donde se generan topologías aleatorias con un número variable de nodos.

6.2.1. Medición de tiempos de ejecución

Todas las pruebas se han realizado utilizando grafos generados con el modelo de Erdős–Rényi, con una probabilidad de conexión fija p entre pares de nodos. Este valor permite generar redes suficientemente densas como para garantizar múltiples caminos potenciales entre el nodo origen y el nodo destino, sin provocar una explosión combinatoria en el número de rutas posibles.

En este caso vamos a medir el tiempo necesario desde el momento en que se lanza la función de selección de ruta hasta que se obtiene un resultado (ya sea un camino válido o la conclusión de que no existe uno que cumpla los requisitos). La medición se realiza utilizando la librería estándar `time` de Python:

```
start_time = time.perf_counter()
result = route_selection_option1(G, source_node, destination_node,
    max_latency, required_bw)
end_time = time.perf_counter()
elapsed_time = end_time - start_time
```

Este tiempo se anota para cada ejecución del algoritmo sobre una topología específica. Posteriormente, se repite la prueba con distintos tamaños de grafo, incrementando progresivamente el número de nodos. Se emplea la misma semilla aleatoria para garantizar la reproducibilidad de los resultados.

6.2.2. Comparativa en función del número de nodos

Para visualizar la evolución del tiempo de ejecución en función del tamaño de la red, se ha construido la siguiente gráfica. El eje horizontal representa el número de nodos del grafo generado, mientras que el eje vertical muestra el tiempo de ejecución del algoritmo (en segundos).

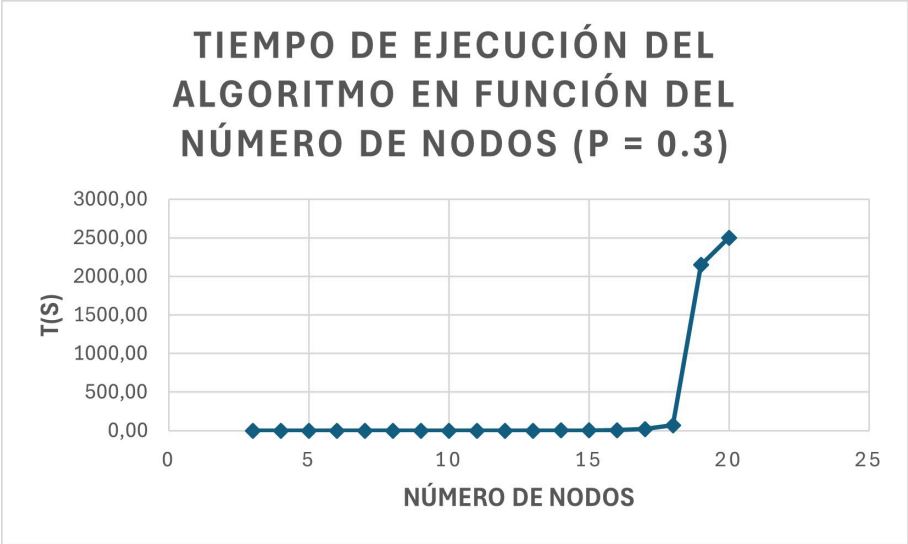


Fig. 6.5. Tiempo de ejecución del algoritmo en función del número de nodos (p = 0.3).

Como se observa en la Figura 6.5, el tiempo de procesamiento se mantiene bajo para valores pequeños de n, pero a partir de cierto umbral (en este caso, alrededor de $n = 18$), se produce un crecimiento abrupto. Este fenómeno está relacionado con el número de caminos posibles entre dos nodos, que en grafos densos puede crecer de manera exponencial.

La siguiente figura muestra el comportamiento bajo condiciones idénticas pero con una menor probabilidad de enlace, concretamente $p = 0{,}2$.

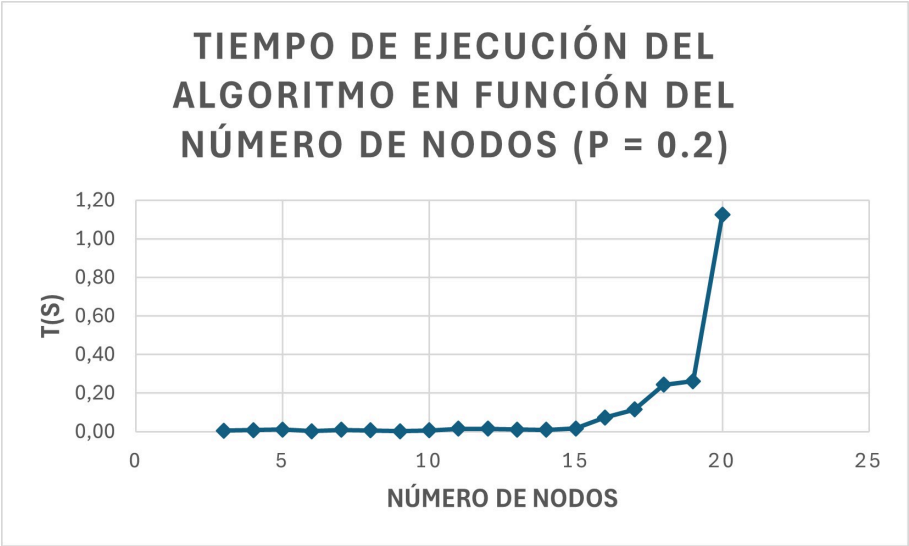


Fig. 6.6. Tiempo de ejecución del algoritmo en función del número de nodos (p = 0.2).

Como puede apreciarse, la pendiente de crecimiento es mucho más suave, ya que la menor densidad de enlaces reduce la cantidad de rutas candidatas y, por tanto, la carga computacional del algoritmo.

6.2.3. Discusión sobre la escalabilidad y eficiencia computacional

Los resultados empíricos obtenidos reflejan una clara correlación entre la densidad de enlaces de la red y el tiempo de ejecución del algoritmo. A medida que la red crece en número de nodos y densidad, el número de rutas posibles entre dos extremos también aumenta, lo cual impacta directamente en la eficiencia del sistema.

De forma teórica, el número de enlaces posibles en un grafo no dirigido con n nodos está dado por la fórmula combinatoria [75]:

$$E_{max} = \binom{n}{2} = \frac{n(n-1)}{2}$$

Y el número esperado de enlaces en un grafo Erdős–Rényi con probabilidad p es:

$$\mathbb{E}[E] = p \cdot \binom{n}{2}$$

Este crecimiento cuadrático en el número de enlaces implica una potencial explosión en la cantidad de rutas simples entre dos nodos, ya que el número de caminos posibles crece rápidamente con la conectividad. Esta explosión combinatoria hace que, aunque el algoritmo sea correcto, pueda volverse ineficiente para topologías densas y grandes.

Por ello, la elección de la probabilidad p representa un compromiso clave entre conectividad (disponibilidad de caminos) y eficiencia computacional (tiempo de cálculo).

Análisis en función de la probabilidad de enlace

Para completar el estudio, se ha realizado un análisis adicional en el que se fija el número de nodos en $n = 10$ y se varía la probabilidad de enlace p desde 0,1 hasta 1,0 en pasos de 0,1. El objetivo es observar cómo afecta la densidad de la red al rendimiento del algoritmo, manteniendo constante el tamaño de la topología.

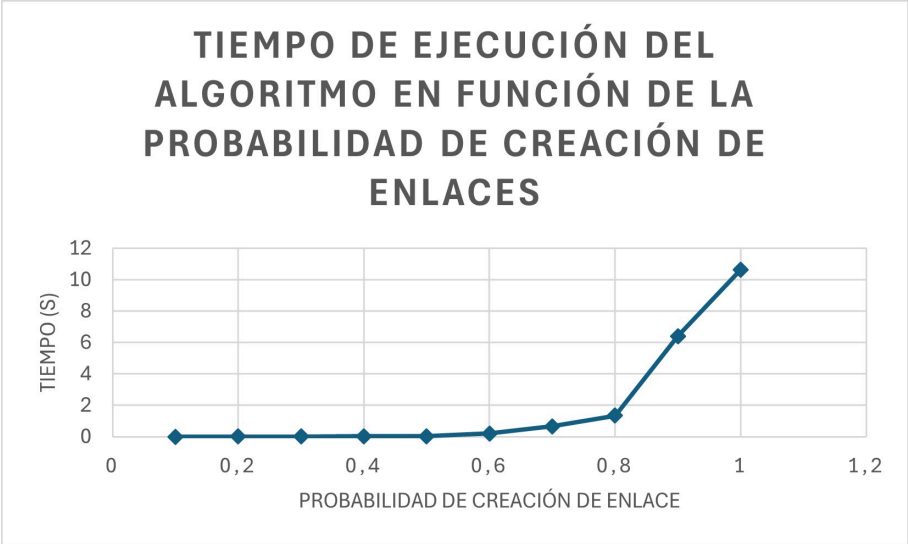


Fig. 6.7. Tiempo de ejecución del algoritmo en función de la probabilidad de enlace (n = 10).

La Figura 6.7 muestra cómo el tiempo de ejecución tiende a incrementarse conforme aumenta la densidad de la red. Esto se debe a que con más enlaces disponibles, el número de caminos viables entre origen y destino también crece, lo que aumenta la carga del proceso de validación de cada ruta (comprobación de latencia y ancho de banda).

Este comportamiento justifica la necesidad de considerar técnicas de poda, heurísticas o algoritmos más eficientes en escenarios reales de gran escala.

6.2.4. Discusión de resultados y lecciones aprendidas

Robustez del sistema ante casos borde

A lo largo de las pruebas realizadas, el sistema ha demostrado un comportamiento estable incluso ante condiciones límite. Por ejemplo, cuando se alcanzan los máximos de ocupación de la red y no existen rutas viables, el algoritmo responde correctamente indicando la imposibilidad de asignar el servicio, sin provocar fallos de ejecución ni bloqueos en el sistema. Además, ante operaciones de descarte, el sistema reacciona limpiamente liberando los recursos y actualizando tanto la red como la base de datos, lo que garantiza la coherencia de estado incluso tras múltiples ciclos de configuración y desconfiguración.

Ajuste del diseño a las condiciones simuladas

El diseño modular planteado ha permitido adaptar el comportamiento del sistema a distintas condiciones de carga. La separación entre componentes —como el selector de rutas, el watchdog y el configurador remoto— ha facilitado tanto la monitorización como la ejecución de pruebas independientes. Asimismo, se ha comprobado que el algoritmo responde adecuadamente cuando varía la latencia máxima permitida o el ancho de banda requerido por los servicios, seleccionando caminos coherentes con los objetivos de calidad definidos.

No obstante, tal y como se discutía en la sección 4.3.5, la estrategia actual de explorar exhaustivamente todos los caminos simples entre origen y destino presenta ciertas limitaciones de escalabilidad. Estas se evidencian especialmente en los resultados obtenidos durante la simulación de carga, donde el tiempo de cómputo crece rápidamente con el número de nodos y enlaces de la red.

Posibilidades de mejora futura

Aunque el sistema ha demostrado ser funcional y robusto, los resultados invitan a explorar nuevas líneas de optimización. Como ya se anticipaba en el apartado 4.3.5, una de las prioridades de mejora reside en la eficiencia computacional. En concreto, se podrían incorporar:

- **Técnicas de poda** que limiten la exploración a caminos prometedores, evitando evaluaciones redundantes.
- **Heurísticas adaptativas** que prioricen rutas en función de la fiabilidad histórica o del tipo de tráfico.
- **Mecanismos de clasificación** para diferenciar entre servicios críticos y no críticos, ajustando los requisitos y criterios de selección.
- **Modelos de aprendizaje automático**, capaces de predecir rutas viables a partir de datos previos, reduciendo la necesidad de cálculo exhaustivo.

Estas ideas se abordarán con mayor profundidad en el capítulo de *Future Work*.

7. CONCLUSIONES

El desarrollo de este Trabajo de Fin de Grado ha dado lugar a un sistema funcional para la provisión automatizada de servicios E2E sobre redes deterministas, siguiendo los principios de la arquitectura DetNet. La solución propuesta se basa en un diseño modular, orientado a la separación de responsabilidades entre los distintos componentes: selección de ruta, monitorización de servicios, configuración remota de nodos y trazabilidad del sistema.

Entre los principales logros alcanzados destaca la implementación de un algoritmo de selección de rutas capaz de identificar caminos viables según requisitos de calidad de servicio (QoS), priorizando aquellos con mayor latencia dentro de un umbral aceptable. Este enfoque ha demostrado ser efectivo tanto en escenarios controlados como en simulaciones con topologías aleatorias, donde se ha verificado su robustez y comportamiento predecible ante situaciones límite, como la saturación de recursos o la recepción de peticiones de descarte.

Asimismo, el sistema incluye un mecanismo de configuración automática de nodos basado en SSH/SCP, con modificación dinámica de archivos JSON distribuidos, lo que permite adaptar la red en tiempo real a las nuevas solicitudes sin intervención manual. El proceso inverso de desconfiguración también ha sido abordado, cerrando el ciclo completo de gestión del servicio.

Desde el punto de vista técnico, las decisiones adoptadas —como el uso de grafos dirigidos con atributos enriquecidos, la persistencia en `database.csv`, o la segmentación en módulos independientes— han facilitado tanto la implementación como las pruebas del sistema. No obstante, se ha identificado que la escalabilidad del algoritmo actual presenta limitaciones computacionales a partir de cierto tamaño de red, lo que abre oportunidades de mejora que se discuten en los capítulos finales.

En conjunto, este TFG ha aportado una solución práctica y extensible, alineada con los requisitos de las redes deterministas modernas, y sienta las bases para futuras investigaciones orientadas a la optimización inteligente de rutas y la gestión dinámica de servicios bajo condiciones de carga variables.

8. PLANIFICACIÓN

El proyecto comenzó en **octubre de 2024** y concluyó en **junio de 2025**. Siguiendo la estructura del TFG, las fases se distribuyeron así (véase Fig. 8.1):

1. **Estudio previo y estado del arte** (Oct.–Nov. 2024)
2. **Análisis y especificación** (Nov.–Dic. 2024)
3. **Diseño de la solución** (Ene. 2025)
4. **Implementación** (Feb.–Abr. 2025)
5. **Resultados y evaluación** (Abr.–May. 2025)
6. **Redacción y revisión** (Nov. 2024–Jun. 2025)

Durante el desarrollo se celebraron **10 reuniones** con el tutor (10 h totales). El esfuerzo técnico se resume en unas 2 h/día × 30 d/mes × 8 meses ≈ 490 h de trabajo.

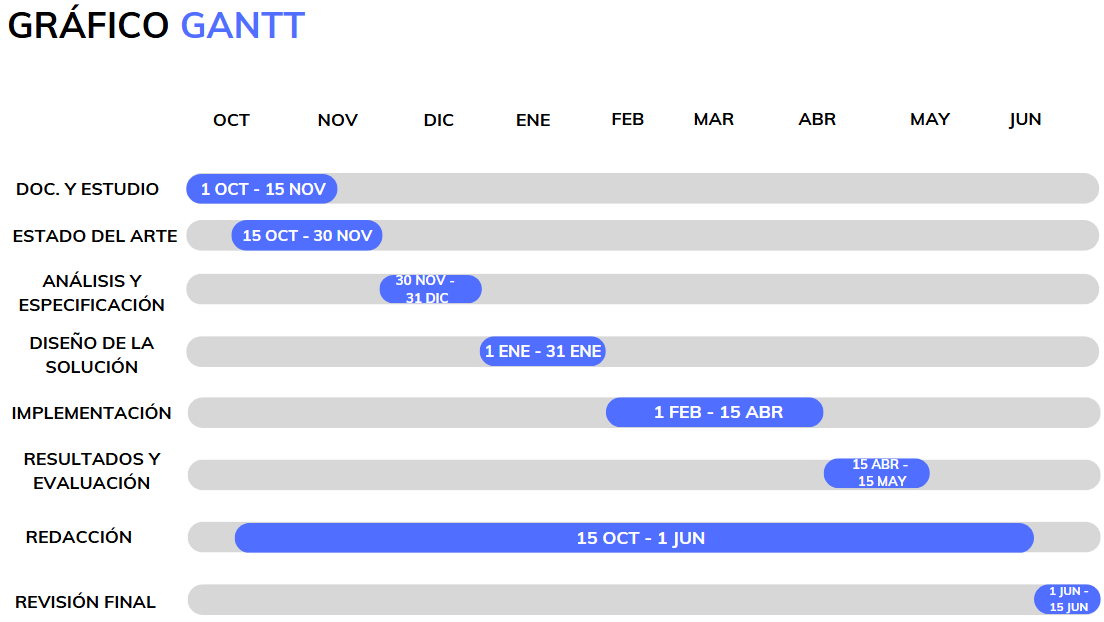


Fig. 8.1. Diagrama de Gantt del proyecto

9. MARCO REGULADOR

En este capítulo se repasan los principales marcos normativos, técnicos y de propiedad intelectual que condicionan el desarrollo del plano de control Wi-Fi dentro del proyecto *PREDICT-6G* y, en particular, la solución *E2E Configurator* presentada en este TFG.

9.1. Legislación aplicable

Ámbito geográfico

El proyecto se financia con fondos europeos y se ejecuta en entidades españolas, por lo que debe acatar tanto la normativa de la Unión Europea como su transposición en España.

Leyes y directivas relevantes

- **Directiva 2014/53/UE (RED)**: Requisitos esenciales de seguridad y compatibilidad electromagnética para equipos de radio, aplicables a dispositivos Wi-Fi en la banda regulada de 2,4–5 GHz. [76]
- **Reglamento (UE) 2016/679 (GDPR)**: Protección de datos personales. El plano de control recopila identificadores de servicio (*service IDs*) y debe garantizar principios de minimización y finalidad. [77]
- **Ley 9/2014, General de Telecomunicaciones**: Régimen de explotación de redes, uso del espectro y despliegue de infraestructuras en España.[78]
- **Reglamento de Ejecución (UE) 2017/1480**: Condiciones de uso armonizado del espectro en la Unión.[79]

9.2. Normativa técnica

Estándares IEEE (Wi-Fi & Determinismo)

- **IEEE 802.11ax / 802.11be**: Capas PHY/MAC de alta eficiencia (Wi-Fi 6/7) empleadas por los APs que gestionará el *Configurator*.[80], [81]
- **IEEE 802.1AS**: *Precision Time Protocol* para sincronización sub-µs, indispensable para el *Time-Aware Shaper* descrito en la subsection 3.1.2.[82]
- **IEEE 802.1Qbv**: Define el *Time-Aware Shaper* (TAS).[19]
- **IEEE 802.1Q/802.1Qaz**: Etiquetado VLAN y prioridades (*QoS*). [1], [83]

Especificaciones IETF DetNet

- **RFC 8655**: Arquitectura DetNet end-to-end.[2]
- **RFC 9055**: Consideraciones de seguridad DetNet (ataques de retardo, *man-in-the-middle*, etc.).[7]
- **RFC 9566**: Funciones PREOF sobre MPLS/UDP utilizadas por los routers open-source basados en XDP mencionados en subsection 3.4.4.[10]

Regulaciones ETSI

- **EN 300 328** (2,4 GHz) y **EN 301 893** (5 GHz): Límites de potencia y ocupación de canal para equipos WLAN en la UE.[84], [85]

9.3. Ética y responsabilidad

- **Uso de software de terceros**. Se reutilizó código de *Intel ixgbe* únicamente para validación en laboratorio; no se redistribuye, respetando su licencia BSD.
- **Protección de datos**. Las trazas se anonimizaron y se controló el acceso al servidor según el GDPR (section 9.1).
- **Responsabilidad profesional**. Se adoptan los principios del Código Deontológico del Colegio Oficial de Ingenieros de Telecomunicación (seguridad y continuidad de servicio).

9.4. Propiedad intelectual

- **Código propio**. Todo el software del *E2E Configurator* es original y se libera bajo licencia MIT, fomentando la colaboración académica.
- **Patentes**. No se detectan invenciones patentables; los algoritmos de selección de ruta se publican como obra derivada académica (§ 4.3).
- **Derechos de autor**. Diagramas y figuras están registrados conforme a la Ley 1/1996 de Propiedad Intelectual.

En conjunto, este marco regulador asegura que el desarrollo se ajusta a la norma técnica (IEEE/IETF), la legalidad europea-española y los principios ético-profesionales, garantizando la futura transferencia y explotación de la solución.

10. ENTORNO SOCIO-ECONÓMICO

Este capítulo analiza, por un lado, el coste de elaboración del TFG y, por otro, el impacto económico, social, medio-ambiental y ético que puede derivarse de la aplicación práctica del *E2E Configurator* para redes deterministas basadas en DetNet/TSN (aplicaciones descritas en Sección 2.1.5).

10.1. Presupuesto de elaboración del TFG

Recursos humanos

El esfuerzo total estimado es el siguiente:

Rol	Tarifa (€/h)	Horas	Coste (€)
Autor (estudiante)	9	490	4 410
Tutor académico	15	120	1 800
		Total	**6 210**

Recursos materiales

- **Equipo personal**: portátil propio (amortizado, sin cargo).
- **VM IMDEA Networks**: uso gratuito proporcionado por el proyecto PREDICT-6G.
- **Software**: Python 3, `networkx`, LaTeX— todos open-source.

Coste total del TFG: *6 210 €*. No se requieren gastos extra en licencias ni hardware.

10.2. Impacto socio-económico esperado

10.2.1. Impacto económico e industrial

- **Industria 4.0**. La integración de redes deterministas en los entornos de fabricación permite ejecutar control de movimiento en robots colaborativos y sistemas de transporte interno con latencias sub-milisegundo, eliminando retardos imprevisibles y paradas de línea debidas a colisiones de tráfico. Los casos piloto descritos por UNIR indican que la migración de Ethernet convencional a infraestructuras TSN/DetNet puede recortar los tiempos de ciclo y las paradas imprevistas hasta en un

15 %, traduciéndose en aumentos directos de productividad y menor desperdicio de material [23]. Además, disponer de un configurador *E2E* acelera la reconfiguración de células flexibles: al automatizar rutas y ventanas temporales la línea puede adaptarse de forma casi inmediata a nuevos lotes o variantes de producto, reforzando el paradigma *mass customization*.

- **Telecomunicaciones**. En las operadoras, la provisión manual de servicios con QoS garantizada suele alargarse por las tareas de diseño, pruebas y activación que hay que repetir equipo a equipo. El configurador propuesto automatiza todo el proceso —desde el cálculo de la ruta determinista hasta la reserva de recursos— y, con ello, reduce tanto el *time-to-market* como los costes de operación asociados a visitas presenciales y ajustes reiterados. A medio plazo, la posibilidad de ofrecer *slices* deterministas "bajo demanda" abre nuevas oportunidades B2B (por ejemplo, enlaces de baja latencia para gemelos digitales industriales o servicios de realidad extendida cooperativa), incrementando el valor de la cartera comercial del operador.

- **Vehículos conectados / V2X**. Las aplicaciones cooperativas (alertas de peligro, platooning, conducción autónoma nivel 4-5) necesitan que los mensajes CAM/CPM se entreguen dentro de ventanas de 50–100 ms y con fiabilidad superior al 99,999 %. El estudio de Sharma, Majumdar y Sivalingam demuestra que la protección 1+1 de DetNet, combinada con algoritmos de planificación de ruta, mejora en un 40 % la entrega puntual de estos mensajes frente a IP best-effort [43]. El configurador facilita esa protección al automatizar PREOF/FRER y seleccionar rutas disjuntas con reserva de ancho de banda; así se habilita la integración de nodos carretera-nube-vehículo en proyectos de infraestructura inteligente.

10.2.2. Impacto social y de accesibilidad

- **Ciudades inteligentes**. En los planes de transformación urbana, la combinación de cámaras 4K y sensores conectados sobre redes deterministas puede permite que los centros de control reciban vídeo y telemetría prácticamente en tiempo real. Esto se traduce en una gestión más ágil del tráfico, por ejemplo, ajuste dinámico de semáforos o avisos tempranos a los servicios de emergencia sin necesidad de desplegar costosos enlaces dedicados; basta con priorizar de forma estricta el tráfico crítico (alarmas, control semafórico) sobre los flujos de menor prioridad.

- **Reducción de brecha digital**. En muchas zonas rurales la conectividad depende de redes Wi-Fi comunitarias, donde la latencia y la variabilidad de los enlaces penalizan aplicaciones interactivas (p. ej. vídeo-clases o teleasistencia). Equipar los puntos de acceso Wi-Fi 6 con políticas deterministas de reserva de recursos—tal y como habilita el configurador E2E—estabiliza estos servicios al garantizar ancho de banda y retardos acotados. Gracias a la generación automática de perfiles reproducibles incluso en hardware económico, la propuesta acerca la calidad de ex-

periencia de los usuarios rurales a la disponible en entornos urbanos sin necesidad de enlaces dedicados.

- **Salud digital**. En aplicaciones de telemedicina, especialmente en telecirugía, es imprescindible mantener la latencia y el *jitter* muy bajos para garantizar la precisión y la seguridad de la intervención. Con la automatización E2E propuesta, un hospital puede aprovisionar rutas deterministas sobre la red troncal existente (IP/MPLS junto con dominios TSN en los quirófanos), sin necesidad de circuitos dedicados. De este modo se acelera la disponibilidad de servicios de cirugía remota y monitorización continua con fiabilidad clínica.

10.2.3. Sostenibilidad y medio ambiente

La optimización conjunta de latencia y ancho de banda (Sección 3.1.2) reduce sobredimensionamiento de enlaces, lo que implica menor consumo energético en la capa de acceso y en los centros de datos [86]. Además, el plano de control permite apagar enlaces redundantes fuera de horario pico, contribuyendo a los ODS 9 y 12 de la ONU.

10.2.4. Ética y responsabilidad

- Cumplimiento del GDPR en la anonimización de *logs* (Cap. 9).
- Principios de código abierto: el software se libera con licencia MIT, fomentando la reutilización académica.
- Riesgo de monopolio: al ser una herramienta abierta, se mitiga la dependencia de proveedores únicos de equipamiento determinista.

10.3. Conclusión

El impacto directo del configurador **no se limita** a una mejora técnica; puede traducirse en ganancias económicas (menor OPEX), beneficios sociales (acceso estable) y reducción de consumo energético. Aunque el efecto real dependerá de su adopción industrial, el análisis indica un potencial positivo y alineado con la transformación digital descrita en la Sección 2.1.5.

11. TRABAJOS FUTUROS

Aunque el sistema desarrollado en este Trabajo de Fin de Grado cumple con los objetivos inicialmente planteados y proporciona una solución funcional para la configuración automática de servicios E2E sobre redes deterministas, existen diversas líneas de mejora y expansión que pueden abordarse en trabajos futuros. A continuación, se recogen las más relevantes, tanto desde el punto de vista técnico como de investigación aplicada.

Optimización del algoritmo de selección

Tal y como se discutió en los apartados 4.3.5 y 6.2, el algoritmo actual se basa en explorar todos los caminos simples entre el origen y el destino, lo cual puede derivar en una complejidad computacional elevada conforme crece el tamaño y la densidad de la red. Para mejorar su escalabilidad, se propone:

- Implementar técnicas de poda temprana que descarten caminos inviables en etapas iniciales del recorrido.
- Aplicar heurísticas basadas en la probabilidad de viabilidad (por ejemplo, evitando caminos con baja capacidad residual).
- Reemplazar el enfoque de exploración exhaustiva por algoritmos más eficientes, como Dijkstra modificado con restricciones múltiples o k-shortest paths con filtrado por QoS.

Priorización de servicios y perfiles de tráfico

Actualmente, todos los servicios entrantes son tratados de forma homogénea. Una mejora sustancial sería permitir la clasificación de los servicios según su criticidad (e.g., tráfico de control industrial vs. streaming multimedia) e introducir mecanismos de priorización o reserva anticipada de recursos. Esta mejora podría incluir:

- Etiquetado de servicios por nivel de prioridad y decisión adaptativa del algoritmo de selección.
- Integración con políticas de calidad de servicio jerarquizadas a nivel de red.

Integración real con routers DetNet

Una línea natural de continuidad consiste en adaptar el sistema para su integración completa con dispositivos de red reales que soporten funcionalidades DetNet, ya sea me-

diante software como Open vSwitch con soporte para TSN o routers comerciales con APIs abiertas. Esto implicaría:

- Estandarizar los archivos de configuración (por ejemplo, en formato YANG o Netconf).
- Establecer comunicación bidireccional para conocer el estado real de los flujos configurados.
- Validar el sistema en entornos de laboratorio con dispositivos físicos o virtualizados.

Aprendizaje automático y predicción de rutas

Otra línea especialmente prometedora es el uso de técnicas de inteligencia artificial para mejorar la toma de decisiones. A partir del histórico de rutas, características de tráfico y disponibilidad de recursos, se podrían entrenar modelos capaces de predecir rutas viables o detectar cuellos de botella de forma proactiva.

Entre las técnicas aplicables destacan:

- Árboles de decisión o redes neuronales ligeras para clasificación de rutas.
- Algoritmos de refuerzo para aprendizaje en entornos dinámicos.
- Sistemas de recomendación de rutas basados en análisis de patrones históricos.

Nuevas funcionalidades y adaptabilidad

El sistema podría ampliarse con funcionalidades orientadas a escenarios más complejos o específicos, como:

- Soporte para caminos redundantes o configuraciones multicamino (multi-path routing) para tolerancia a fallos.
- Visualización dinámica del estado de la red y de los servicios activos mediante dashboards en tiempo real.
- Extensión de la lógica del watchdog para detectar anomalías o caídas de nodos, no solo nuevos servicios.

Evaluación en redes a gran escala

Hasta el momento, las pruebas se han realizado en escenarios controlados y topologías aleatorias de tamaño moderado. Un paso clave hacia la validación completa sería desplegar el sistema en entornos simulados a gran escala (e.g., Mininet, Containernet) o incluso en infraestructuras federadas como Fed4FIRE o testbeds académicos, donde se puedan simular centenares de nodos bajo condiciones realistas.

Aplicaciones futuras

Finalmente, este sistema podría adaptarse a otros contextos más allá de DetNet, como:

- Redes industriales críticas (IEC/IEEE Time-Sensitive Networking).
- Vehículos autónomos conectados (V2X) donde la latencia y fiabilidad son cruciales.
- Reducción de latencia en redes 5G para servicios de misión crítica.
- Entornos de edge computing con orquestación de flujos deterministas entre nodos de borde.

En conjunto, las mejoras aquí descritas sientan las bases para convertir este sistema en una solución robusta, inteligente y plenamente aplicable en escenarios reales de redes deterministas. Estas líneas abren oportunidades no solo para proyectos de ingeniería, sino también para trabajos de investigación en áreas emergentes de las telecomunicaciones.

BIBLIOGRAFÍA

[1] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks*, 2022. DOI: 10.1109/IEEESTD.2022.10004498.

[2] N. Finn, P. Thubert, B. Varga y J. Farkas, *Deterministic Networking Architecture*, RFC 8655 (Standards Track), Internet Engineering Task Force, oct. de 2019. DOI: 10.17487/RFC8655. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc8655.

[3] N. D. Blog. "Understanding the Need for Time-Sensitive Networking for Critical Applications." (2022), [En línea]. Disponible en: https://developer.nvidia.com/blog/understanding-the-need-for-time-sensitive-networking-for-critical-applications.

[4] C. Lefelhocz, B. Lyles, S. Shenker y L. Zhang, "Congestion Control for Best-Effort Service: Why We Need a New Paradigm," *IEEE Network*, vol. 10, n.º 1, pp. 10-19, ene. de 1996. DOI: 10.1109/65.484227.

[5] IETF DetNet Working Group, *Deterministic Networking (DetNet) Working Group Charter*, Internet Engineering Task Force, Datatracker, Version 00-00, last updated 28 Aug 2015. Available: https://datatracker.ietf.org/doc/charter-ietf-detnet/00-00/, ago. de 2015.

[6] International Organization for Standardization, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*, 1994.

[7] E. Grossman, T. Mizrahi y A. J. Hacker, *Deterministic Networking (DetNet) Security Considerations*, Request for Comments 9055, jun. de 2021. DOI: 10.17487/RFC9055. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9055.

[8] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks—Audio Video Bridging (AVB) Systems*, 2011. DOI: 10.1109/IEEESTD.2011.6032690. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/6032690.

[9] V. Addanki y L. Iannone, "Moving a step forward in the quest for Deterministic Networks (DetNet)," en *2020 IFIP Networking Conference (Networking)*, IEEE, 2020, pp. 458-466.

[10] D. Brungard, G. Mirsky y X. Li, *Deterministic Networking (DetNet) Packet Replication, Elimination, and Ordering Functions (PREOF) via MPLS over UDP/IP*, Request for Comments 9566, abr. de 2024. DOI: 10.17487/RFC9566. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9566.

[11] D. Brungard, N. Finn y J. Farkas, *Deterministic Networking (DetNet) Data Plane: MPLS*, Request for Comments 8964, ene. de 2021. DOI: 10.17487/RFC8964. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc8964.

[12] D. Brungard, N. Finn y J. Farkas, *Deterministic Networking (DetNet) Data Plane: IP over TSN*, Request for Comments 9023, jun. de 2021. DOI: 10.17487/RFC9023. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9023.

[13] D. Brungard, N. Finn y J. Farkas, *Deterministic Networking (DetNet) Data Plane: MPLS over UDP/IP*, Request for Comments 9025, abr. de 2021. DOI: 10.17487/RFC9025. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9025.

[14] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks— Frame Replication and Elimination for Reliability*, 2017. DOI: 10.1109/IEEESTD.2017.8091139.

[15] IEEE Instrumentation and Measurement Society, *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2019. [En línea]. Disponible en: https://standards.ieee.org/standard/1588-2019.html.

[16] D. O. Awduche, A. Y. Chiu, A. Elwalid, I. Widjaja y X. Xiao, *Overview and Principles of Internet Traffic Engineering*, Request for Comments 9522, ene. de 2023. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9522.

[17] B. Braden, L. Zhang, S. Berson, S. Herzog y S. Jamin, *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*, Request for Comments 2205, sep. de 1997. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc2205.

[18] M. Boiger e I. 8. W. Group, *Time-Aware Shaper*, IEEE 802.1 Public Document, Version 01, jul. de 2012. [En línea]. Disponible en: https://www.ieee802.org/1/files/public/docs2012/bv-boiger-time-aware-shaper-0712-v01.pdf.

[19] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic*, 2015. DOI: 10.1109/IEEESTD.2015.7295477. [En línea]. Disponible en: https://standards.ieee.org/standard/802_1Qbv-2015.html.

[20] A. L. Zumeta, "Análisis y diseño de un sistema de sincronización y scheduling para Time-Sensitive Networking (TSN)," Sección 2.2.2: Scheduling en TSN, Tesis de mtría., Universidad Miguel Hernández de Elche, Elche, España, 2020. [En línea]. Disponible en: https://dspace.umh.es/bitstream/11000/31550/1/TESIS%20SF%20Ana%20Larra%C3%B1aga%20Zumeta.pdf.

[21] J. Farkas, B. Varga, S. Bryant et al., *Deterministic Networking (DetNet) Bounded Latency*, Request for Comments 9320, 2022. DOI: 10.17487/RFC9320. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9320.

[22] D. O. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan y G. Swallow, *RSVP-TE: Extensions to RSVP for LSP Tunnels*, Request for Comments 3209, dic. de 2001. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc3209.

[23] UNIR. "La industria 4.0, la transformación digital de las organizaciones industriales." Consultado el 15 de mayo de 2025. (2021), [En línea]. Disponible en: https://www.unir.net/revista/ingenieria/que-es-industria-4-0/.

[24] P. Szilagyi, L. M. Contreras, D. Rico-Menéndez, P. Giardina y A. de la Oliva, "6G Architecture for Enabling Predictable, Reliable and Deterministic Networks: the PREDICT-6G Case," en *IEEE Wireless Communications and Networking Conference (WCNC)*, 2024, pp. 1-6.

[25] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, 2018. [En línea]. Disponible en: https://standards.ieee.org/ieee/802.1Qcc/5784/.

[26] VMware. "What is Software-Defined Networking (SDN)?" (2025), [En línea]. Disponible en: https://www.vmware.com/topics/software-defined-networking.

[27] L. Silva, B. Santos, J. Suto, L. Tomaś y E. Monteiro, "On the Adequacy of SDN and TSN for Industry 4.0," en *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, 2019, pp. 43-51.

[28] OpenDaylight Project. "NETCONF User Guide." (2025), [En línea]. Disponible en: https://docs.opendaylight.org/projects/netconf/en/latest/user-guide.html.

[29] O. Project, *NETCONF Integration in ONOS*, 2025. [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/NETCONF.

[30] R. Enns, M. Bjorklund, J. Schoenwaelder y A. Bierman, *Network Configuration Protocol (NETCONF)*, RFC 6241, 2011. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc6241.

[31] M. Bjorklund, *The YANG 1.1 Data Modeling Language*, RFC 7950, 2016. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc7950.

[32] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 30: YANG Data Model*, 2018. [En línea]. Disponible en: https://standards.ieee.org/ieee/802.1Qcp/6238/.

[33] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 33: YANG Data Model for Connectivity Fault Management*, 2020. [En línea]. Disponible en: https://standards.ieee.org/ieee/802.1Qcx/7135/.

[34] A. A. Group. "What Is OPC UA TSN for Industrial Networking?" (2020), [En línea]. Disponible en: https://www.arcweb.com/industry-best-practices/what-opc-ua-tsn-industrial-networking.

[35] T. Kobzan, T. Benesl y P. Michal, "Configuration Solution for TSN-Based Industrial Networks Utilizing SDN and OPC UA," en *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2020, pp. 1335-1338.

[36] S. Arestova, T. Sander, T. Streichert y G. Bauer, "A Service-Oriented Real-Time Communication Scheme for AUTOSAR Adaptive Using OPC UA and TSN," *Sensors*, vol. 21, n.º 7, p. 2337, 2021.

[37] O. S. A. D. L. (OSADL). "OPC UA PubSub over TSN." (2020), [En línea]. Disponible en: https://www.osadl.org/OPC-UA-PubSub-over-TSN.opcua-tsn.0.html.

[38] W. Sim, B. Song, J. Shin y T. Kim, "Data Distribution Service Converter Based on the Open Platform Communications Unified Architecture Publish–Subscribe Protocol," en *Electronics*, 2021. [En línea]. Disponible en: https://www.researchgate.net/figure/OPC-UA-PubSub-communication-model-3_fig2_355374149.

[39] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959. DOI: 10.1007/BF01386390. [En línea]. Disponible en: https://link.springer.com/article/10.1007/BF01386390.

[40] S.-H. Chang, H. Chen y B.-C. Cheng, "Time-predictable routing algorithm for Time-Sensitive Networking: Schedulable guarantee of Time-Triggered streams," *Computer Communications*, vol. 176, pp. 1-12, 2021. DOI: 10.1016/j.comcom.2021.05.003. [En línea]. Disponible en: https://doi.org/10.1016/j.comcom.2021.05.003.

[41] J.-Y. Huang, M.-H. Hsu y C.-A. Shen, "A Novel Routing Algorithm for the Acceleration of Flow Scheduling in Time-Sensitive Networks," *Sensors*, vol. 20, n.º 21, p. 6400, 2020. DOI: 10.3390/s20216400. [En línea]. Disponible en: https://www.mdpi.com/1424-8220/20/21/6400.

[42] D. Eppstein, "Finding the k Shortest Paths," *SIAM Journal on Computing*, vol. 28, n.º 2, pp. 652-673, 1998. DOI: 10.1137/S0097539795290477. [En línea]. Disponible en: https://doi.org/10.1137/S0097539795290477.

[43] G. P. Sharma, A. Majumdar y K. M. Sivalingam, "Routing and Scheduling for 1+1 Protected DetNet Flows," *Computer Communications*, vol. 181, pp. 67-81, 2022.

[44] P. E. Hart, N. J. Nilsson y B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, n.º 2, pp. 100-107, 1968. DOI: 10.1109/TSSC.1968.300136.

[45] D. Delling y D. Wagner, "Time-Dependent Route Planning," en *Robust and Online Large-Scale Optimization*, ép. Lecture Notes in Computer Science, R. K. Ahuja, R. H. Möhring y C. D. Zaroliagis, eds., vol. 5868, Springer, 2009, pp. 207-230. DOI: 10.1007/978-3-642-05465-5_8. [En línea]. Disponible en: https://doi.org/10.1007/978-3-642-05465-5_8.

[46] F. Grajales, J. Barrios y S. Yang, "Adaptations of the A* Algorithm for the Computation of Fastest Paths in Dynamic Networks," en *Proceedings IEEE Intelligent Transportation Systems*, IEEE, 2001, pp. 838-843.

[47] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla y G. Mühl, "ILP-based Joint Routing and Scheduling for Time-Triggered Networks," en *Proc. 25th Int. Conf. on Real-Time Networks and Systems (RTNS)*, 2017, pp. 8-17. DOI: 10.1145/3139258.3139289.

[48] A. Finzi y R. S. Oliver, "General Framework for Routing, Scheduling and Formal Timing Analysis in Deterministic Time-Aware Networks," en *IEEE WFCS*, 2022.

[49] A. Alnajim, S. Salehi y C.-C. Shen, "Incremental Path-Selection and Scheduling for Time-Sensitive Networks," en *IEEE GLOBECOM*, 2019, pp. 1-6. [En línea]. Disponible en: https://www.eecis.udel.edu/~salehi/files/globecomm19_incremental.pdf.

[50] W. Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks," en *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2010, pp. 375-384. DOI: 10.1109/RTSS.2010.25. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/5702246.

[51] G. P. Sharma, A. Majumdar y K. M. Sivalingam, "Routing and Scheduling for 1+1 Protected DetNet Flows," en *RTSHi '21*, 2021. [En línea]. Disponible en: https://gourav-prateek-sharma.github.io/files/publications/1+1_rtsch_detnet.pdf.

[52] A. Finzi y R. S. Oliver, "General Framework for Routing, Scheduling and Formal Timing Analysis in Deterministic Time-Aware Networks," en *IEEE WFCS*, 2022.

[53] Y. Li et al., "Heuristic Routing Algorithms for Time-Sensitive Networks in Smart Factories," *Sensors*, vol. 22, n.º 11, p. 4153, 2022. DOI: 10.3390/s22114153. [En línea]. Disponible en: https://www.mdpi.com/1424-8220/22/11/4153.

[54] S. B. H. Said, M.-T. Thi, M. Kellil y A. Olivereau, "On the Management of TSN Networks in 6G: A Network Digital Twin Approach," en *Proc. 28th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1-7. DOI: 10.1109/ETFA54631.2023.10275568.

[55] ManufacturingTomorrow. "TSN Improves Digital Twins in Manufacturing." (2022), [En línea]. Disponible en: https://www.manufacturingtomorrow.com/news/2022/03/28/tsn-improves-digital-twins-in-manufacturing/18460.

[56] B. Shi, X. Tu, B. Wu e Y. Peng, "Recent Advances in Time-Sensitive Network Configuration Management: A Literature Review," *Journal of Sensor and Actuator Networks*, vol. 12, n.º 4, p. 52, 2023. DOI: 10.3390/jsan12040052. [En línea]. Disponible en: https://www.mdpi.com/2224-2708/12/4/52.

[57] K. Park, S. Sung, H. Kim y J. I. Jung, "Technology Trends and Challenges in SDN and Service Assurance for End-to-End Network Slicing," *Computer Networks*, vol. 234, p. 109 908, 2023. DOI: 10.1016/j.comnet.2023.109908. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S1389128623003535.

[58] S. S. Jazaeri, S. Jabbehdari, P. Asghari y H. H. S. Javadi, "Edge Computing in SDN–IoT Networks: Issues, Challenges and Solutions," *Cluster Computing*, vol. 24, n.º 3, pp. 3173-3192, 2021. DOI: 10.1007/s10586-021-03311-6. [En línea]. Disponible en: https://doi.org/10.1007/s10586-021-03311-6.

[59] I. 8. TSN, *Architecture of Time-Sensitive Network Configuration Management*, Encyclopedia Entry, 2023. [En línea]. Disponible en: https://encyclopedia.pub/entry/46705.

[60] I. 8. W. Group, *Clause 46 Extract — Fully Distributed Model (Draft 2024)*, IEEE 802.1 Public Document, 2024. [En línea]. Disponible en: https://www.ieee802.org/1/files/public/docs2024/dj-seaman-d2-0--comments-I-57-and-I-51-clause-46-1-extract-markup-0124-v01.pdf.

[61] D. Cummings, *802.1Qcc UNI: Multi-Port End Systems—Management Gaps*, IEEE 802.1 Presentation, mar. de 2016. [En línea]. Disponible en: https://www.ieee802.org/1/files/public/docs2016/cc-cummings-multi-port-end-systems-0316-v1.pdf.

[62] T. Corp. "Abstracting Configuration Complexity of Time-Sensitive Networking." (2023), [En línea]. Disponible en: https://tenasys.com/wp-content/uploads/2023/09/TenAsys.WP.TSN_.v7-92423.pdf.

[63] A. Bierman y R. Wilton, "CoRECONF: YANG Library and Protocol Mappings for Constrained RESTful Environments (CoRE)," IETF Constrained RESTful Environments (CoRE) WG, Internet-Draft draft-ietf-core-comi-coreconf-yang-library-11, jun. de 2025, Work in Progress. [En línea]. Disponible en: https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-coreconf-yang-library-11.

[64] M. G. Bhat, P. Ramanathan y B. Lingam, "CORECONF, NETCONF, and RESTCONF: Benchmarking Network Orchestration in Constrained IIoT Devices," *IEEE Internet of Things Journal*, vol. 11, n.º 7, pp. 13 082-13 090, 2024.

[65] K. Zanbouri, M. Noor-A-Rahim, J. John, C. J. Sreenan, H. V. Poor y D. Pesch, *A Comprehensive Survey of Wireless Time-Sensitive Networking (TSN): Architecture, Technologies, Applications, and Open Issues*, arXiv:2312.01204, 2023. [En línea]. Disponible en: https://arxiv.org/abs/2312.01204.

[66] K. M. Shalghum, N. K. Noordin, A. Sali y F. Hashim, "Network Calculus-Based Latency for Time-Triggered Traffic under Flexible Window-Overlapping Scheduling (FWOS) in a Time-Sensitive Network (TSN)," *Applied Sciences*, vol. 11, n.º 9, p. 3896, 2021. DOI: 10.3390/app11093896.

[67] D. WG, *Requirements for Scaling Deterministic Networks*, IETF Internet-Draft (draft-ietf-detnet-scaling-requirements-02), ene. de 2024. [En línea]. Disponible en: https://datatracker.ietf.org/doc/draft-ietf-detnet-scaling-requirements/.

[68] F. Fejes, F. Orosi, B. Varga y J. Farkas, *Lightweight Implementation of Per-packet Service Protection in eBPF/XDP*, Netdev 0x17 Technical Conference on Linux Networking, 2023. eprint: 2312.07152. [En línea]. Disponible en: https://arxiv.org/abs/2312.07152.

[69] P. S. Foundation, *Python Language Reference, version 3.13.0*, https://www.python.org, 2025.

[70] A. Ronacher, *Flask – Web Development, One Drop at a Time*, https://flask.palletsprojects.com/, 2025.

[71] A. Hagberg et al., *NetworkX 3.2.1*, https://networkx.org, 2023.

[72] J. Forcier, *Paramiko 3.4 – SSH2 protocol library*, https://www.paramiko.org, 2023.

[73] M. Sobotta, *python-scp 0.14*, https://pypi.org/project/scp/, 2023.

[74] M. D. Team, *Matplotlib 3.9 documentation*, https://matplotlib.org, 2024.

[75] P. Erdős y A. Rényi, "On Random Graphs I," *Publicationes Mathematicae*, vol. 6, pp. 290-297, 1959.

[76] E. Union, *Directive 2014/53/EU of the European Parliament and of the Council on the harmonisation of the laws of the Member States relating to the making available on the market of radio equipment*, Radio Equipment Directive (RED), 2014.

[77] E. Union, *Regulation (EU) 2016/679 General Data Protection Regulation*, 2016.

[78] G. de España, *Ley 9/2014, de 9 de mayo, General de Telecomunicaciones*, 2014.

[79] E. Union, *Implementing Regulation (EU) 2017/1480 on Technical Specifications and Standardisation of Radio Spectrum Information*, 2017.

[80] I. 8. W. Group, *IEEE Standard for Information Technology—Telecommunications and Information Exchange between Systems LAN/MAN—Part 11: Wireless LAN MAC/PHY Specifications; Amendment: High Efficiency WLAN (802.11ax)*, 2021.

[81] *IEEE Draft Standard 802.11be—Extremely High Throughput*, 2024.

[82] *IEEE Std 802.1AS-2020—Timing and Synchronization for Time-Sensitive Applications*, 2020.

[83] IEEE 802.1 Working Group, *IEEE Standard for Local and Metropolitan Area Networks— Virtual Bridged Local Area Networks— Amendment 23: Enhanced Transmission Selection for Bandwidth Sharing Between Traffic Classes*, 2011. [En línea]. Disponible en: https://standards.ieee.org/standard/802_1Qaz-2011.html.

[84] ETSI, *EN 300 328 V2.2.2—Wideband Transmission Systems; 2.4 GHz Band*, 2019.

[85] ETSI, *EN 301 893 V2.1.1—5 GHz RLAN; Harmonised Standard for Access to Radio Spectrum*, 2017.

[86] D. Bezerra et al., "A Machine Learning-Based Optimization for End-to-End Latency in TSN Networks," *Computer Communications*, vol. 200, pp. 134-145, 2023. DOI: 10.1016/j.comcom.2022.12.015. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0140366422003504.

ANEXO

DECLARACIÓN DE USO DE IA GENERATIVA EN EL TRABAJO DE FIN DE GRADO

He usado IA Generativa en este trabajo

Marca lo que corresponda: SI

SI	NO

Si has marcado SI, completa las siguientes 3 partes de este documento:

Parte 1: reflexión sobre comportamiento ético y responsable

Ten presente que el uso de IA Generativa conlleva unos riesgos y puede generar una serie de consecuencias que afecten a la integridad moral de tu actuación con ella. Por eso, te pedimos que contestes con honestidad a las siguientes preguntas (*Marca lo que corresponda*):

Pregunta		
1. En mi interacción con herramientas de IA Generativa he remitido **datos de carácter sensible** con la debida autorización de los interesados.		
SÍ, he usado estos datos con autorización	NO, he usado estos datos sin autorización	NO, no he usado datos de carácter sensible X
2. En mi interacción con herramientas de IA Generativa he remitido **materiales protegidos por derechos de autor** con la debida autorización de los interesados.		
SÍ, he usado estos materiales con autorización	NO, he usado estos materiales sin autorización	NO, no he usado materiales protegidos X
3. En mi interacción con herramientas de IA Generativa he remitido **datos de carácter personal** con la debida autorización de los interesados.		

SÍ, he usado estos datos con autorización	NO, he usado estos datos sin autorización	NO, no he usado datos de carácter personal X

4. Mi utilización de la herramienta de IA Generativa ha **respetado sus términos de uso**, así como los principios éticos esenciales, no orientándola de manera maliciosa a obtener un resultado inapropiado para el trabajo presentado, es decir, que produzca una impresión o conocimiento contrario a la realidad de los resultados obtenidos, que suplante mi propio trabajo o que pueda resultar en un perjuicio para las personas.

SI X	NO

Si **NO** has contado con la autorización de los interesados en alguna de las preguntas 1, 2 ó 3, explica brevemente el motivo *(por ejemplo, "los materiales estaban protegidos pero permitían su uso para este fin" o "los términos de uso, que se pueden encontrar en esta dirección (…), impiden el uso que he hecho, pero era imprescindible dada la naturaleza del trabajo".*

Parte 2: declaración de uso técnico

Utiliza el siguiente modelo de declaración tantas veces como sea necesario, a fin de reflejar todos los tipos de iteración que has tenido con herramientas de IA Generativa. Incluye un ejemplo por cada tipo de uso realizado donde se indique:

Declaro haber hecho uso del sistema de IA Generativa ChatGPT modelo o4-mini-high para *entender ciertas partes y conceptos del estado del arte.*

Declaro haber hecho uso del sistema de IA Generativa ChatGPT modelo o4-mini-high para *solicitar la revision y rescritura de párrafos los cuales ya había escrito con anterioridad.*

Declaro haber hecho uso del sistema de IA Generativa ChatGPT modelo o4-mini-high para realizar *preguntas que se orientaban a ejemplos de uso y explicación de algunos conceptos*

Declaro haber hecho uso del sistema de IA Generativa ChatGPT modelo o4-mini-high para realizar *busqueda de bibliografía asegurandome de que fuese correcta antes de añadirla.*

Declaro haber hecho uso del sistema de IA Generativa ChatGPT modelo o4-mini-high para *pedir explicaciones sobre fragmentos de código en lenguaje Python.*

Declaro haber hecho uso del sistema de IA Generativa Copilot extensión de VS Code para *autocompletar código en python y realizar comentarios en el código.*

Declaro haber hecho uso del sistema de IA Generativa ChatGPT modelo o4-mini-high para *para solucionar errores de código que no fui capaz de solucionar.*

Parte 3: reflexión sobre utilidad

Por favor, aporta una valoración personal (formato libre) sobre las fortalezas y debilidades que has identificado en el uso de herramientas de IA Generativa en el desarrollo de tu trabajo. Menciona si te ha servido en el proceso de aprendizaje, o en el desarrollo o en la extracción de conclusiones de tu trabajo.

Fortalezas

- **Aprendizaje acelerado**.
  Poder formular preguntas en lenguaje natural del tipo *"¿qué diferencia hay entre PREOF y FRER?"* y recibir al instante una respuesta razonada ha reducido drásticamente el tiempo que, de otro modo, habría invertido rastreando documentación dispersa en Internet.
- **Claridad.**
  El uso de IA me ha ayudado a afianzar conceptos (p. ej. arquitecturas DetNet) antes de profundizar en fuentes académicas formales.
- **Redacción formal.**
  Para apartados de descripción (por ejemplo, "impacto socioeconómico" o "resumen en inglés") la IA ha servido como mejora para la redacción formal.
- **Soporte idiomático y de estilo**.
  Revisar la coherencia terminológica (español ↔ inglés técnico) o pedir versiones en lenguaje más conciso ha sido especialmente útil en la fase de redacción final.

Debilidades

- **Alucinaciones y referencias incompletas**.
  En varias ocasiones el modelo citó artículos que no existen o que no abordaban exactamente el tema solicitado. Esto obliga a contrastar cada respuesta con la bibliografía real, añadiendo una capa de verificación necesaria.
- **Información potencialmente desactualizada**.
  Los estándares IEEE y los RFC evolucionan; un modelo entrenado con datos antiguos puede quedarse corto o errar en números de versiones. He tenido que comprobar manualmente fechas y números de norma.

En mi caso, la IA generativa ha actuado como un **mentor instantáneo**: disponible las 24 h, comprensivo con mi lenguaje cotidiano y capaz de proponer caminos de solución cuando me atascaba. La clave ha sido combinar esa agilidad con un proceso sistemático de verificación. Así, la herramienta no sustituye el estudio riguroso ni la lectura de documentos oficiales, pero sí **optimiza el ciclo aprender-probar-corregir**, haciendo el trabajo más eficiente y, al mismo tiempo, enriqueciendo mi comprensión del tema.