UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"ANÁLISIS COMPARATIVO DE LOS PROTOCOLOS ROUND ROBIN Y REDES NEURONALES RECURRENTES (RNN) PARA BALANCEO DE CARGA EN REDES SDN"

Trabajo de titulación previo a la obtención del título de Ingeniero en Telecomunicaciones

Línea de investigación: Innovación Tecnológica y de productos

AUTOR:

Montenegro Padilla Luis Francisco

DIRECTOR:

Hernán Mauricio Domínguez Limaico

Ibarra – Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	MONTENEGRO PADILLA LUIS FRANCISCO

DATOS DE LA OBRA	
TÍTULO:	Análisis comparativo de protocolos Round Robin y Redes Neuronales Recurrentes (RNN) para balanceo de carga en SDN
AUTOR (ES):	Luis Francisco Montenegro Padilla
FECHA: DD/MM/AAAA	16/07/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR /ASESOR:	MSC. HERNÁN MAURICIO DOMINGUEZ LIMAICO / MSC. JAIME ROBERTO MICHILENA CALDERÓN

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 16 días del mes de Julio de 2026

EL AUTOR:



(Firma)……
Nombre: Luis Francisco Montenegro Padilla

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 16 de Julio de 2026

MSc. Hernán Mauricio Domínguez Limaico
DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:
Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.



Validar únicamente en FirmaEC.
Firmado electrónicamente por:
HERNAN MAURICIO DOMINGUEZ LIMAICO

(f) ……
MSC. Hernán Mauricio Domínguez Limaico
C.C.: 1002379301

DEDICATORIA

Dedico este trabajo de titulación, en primer lugar, a Dios, por guiar mis pasos, brindarme la sabiduría necesaria y permitirme culminar esta etapa de mi vida con salud y fortaleza.

A mi familia, a mi mamá Tatiana, a mi papá Mauricio, a mi hermana Mishell y a mi pequeña sobrina Sofía. Ustedes han sido, son y serán siempre mi motor y mi inspiración. Gracias por ser el refugio donde encuentro la motivación constante para superarme día tras día. Este logro es, en gran medida, de ustedes.

A mi fiel compañero, Duke. Gracias por estar a mi lado en las noches de desvelo y en cada altibajo de este camino. Tu amor incondicional y tu lealtad pura han sido el bálsamo que necesitaba para seguir adelante; tu compañía silenciosa ha sido el regalo más grande en este proceso.

A mi mejor amigo, Chelo. Hoy cumplo aquella promesa que alguna vez hicimos juntos. Aunque la vida nos separó físicamente, te siento presente en cada paso que doy. Gracias por todos los años compartidos; aprendí que un hermano verdadero no conoce de tiempo ni de distancias, pues su esencia perdura para siempre en el corazón. Sé que celebras este triunfo conmigo desde el cielo.

Finalmente, a mis amigos de la carrera y especialmente a mi mejor amiga Joyce. Con ustedes compartí risas, retos y esas "desgracias" que, al final, se convirtieron en las mejores anécdotas. Gracias por el apoyo mutuo y por hacer de este trayecto académico una experiencia humana inolvidable. Son, sin duda, el mejor regalo que me llevo de esta etapa.

Luis Francisco Montenegro Padilla

AGRADECIMIENTO

Mi más profundo agradecimiento a mi familia, quienes han sido el pilar fundamental de mi vida y mi mayor fuente de motivación. Gracias por su amor incondicional y por el apoyo constante que me brindaron para alcanzar esta meta.

A la carrera de Ingeniería en Telecomunicaciones de la Universidad Técnica del Norte, por ser el espacio donde no solo adquirí conocimientos técnicos, sino también la sabiduría necesaria para enfrentar los desafíos de nuestra profesión.

Expreso mi gratitud especial a los ingenieros Mauricio Domínguez y Jaime Michilena. Gracias por su guía experta, su paciencia y por compartir conmigo sus valiosos conocimientos. Su mentoría fue decisiva para la culminación exitosa de este trabajo de titulación.

Finalmente, agradezco profundamente a mis amigos: Orlando, Kevin, Eduardo, Farid, Becker, Alejo, los gemelos Rivera, David, Jordy, Jordan, Heidy, Ariel, Nahim y Zhara. Agradezco cada momento compartido, desde los desafíos y las dificultades hasta las risas y las alegrías. Haber recorrido este camino a su lado hizo que esta etapa universitaria fuera, sin duda, una experiencia inolvidable.

Luis Francisco Montenegro Padilla

RESUMEN EJECUTIVO

El crecimiento del tráfico digital y la complejidad de las infraestructuras de red han evidenciado las limitaciones de los algoritmos de balanceo de carga estáticos frente a entornos dinámicos y multiprotocolo, lo que justifica la necesidad de explorar soluciones adaptativas basadas en inteligencia artificial; en este contexto, el presente trabajo tiene como objetivo analizar el desempeño del balanceo de carga basado en redes neuronales para redes SDN en un entorno controlado de simulación. La metodología empleada sigue un modelo en cascada estructurado en cuatro fases: revisión teórica del estado del arte, diseño e implementación de ambos controladores, Round Robin y una red neuronal bidireccional de memoria a largo y corto plazo (LSTM), sobre la plataforma Mininet con el controlador Ryu y el protocolo OpenFlow 1.3, diseño de diez escenarios de prueba con inyección de tráfico heterogéneo (HTTP, VoIP y streaming), y análisis comparativo de métricas estandarizadas bajo ITU-T Y.1540 y RFC 2544 (latencia IPTD, pérdida de paquetes IPLR, variación de retardo IPDV y throughput). Los resultados demuestran que el controlador Bi-LSTM reduce la desviación estándar de uso de CPU entre servidores de un 43.39% a un 4.18% en tráfico mixto concurrente; contiene la saturación de CPU en un 75.49% frente al 99.95% registrado por Round Robin en escenarios de estrés crítico; sostiene una tasa de pérdida de paquetes del 0.0% donde el método estático alcanza el 1.43%; y mantiene una latencia media de 1.11 ms frente a 1.20 ms en pruebas de estabilidad prolongada. Se concluye que la arquitectura LSTM supera significativamente al protocolo Round Robin en eficiencia, equidad de distribución, resiliencia ante fallos y continuidad de servicio, validando su viabilidad operativa en entornos SDN.

Palabras clave: SDN, balanceo de carga, LSTM, Round Robin, redes neuronales recurrentes, OpenFlow.

ABSTRACT

The growth of digital traffic and the complexity of network infrastructures have exposed the limitations of static load balancing algorithms in dynamic and multiprotocol environments, justifying the need to explore adaptive solutions based on artificial intelligence; in this context, the present work aims to analyze the performance of neural network-based load balancing for SDN networks in a controlled simulation environment. The methodology follows a waterfall model structured in four phases: theoretical review of the state of the art, design and implementation of both controllers —Round Robin and a Bidirectional Long Short-Term Memory neural network (Bi-LSTM)— on the Mininet platform with the Ryu controller and the OpenFlow 1.3 protocol, design of ten test scenarios with heterogeneous traffic injection (HTTP, VoIP, and streaming), and comparative analysis of standardized metrics under ITU-T Y.1540 and RFC 2544 (latency IPTD, packet loss IPLR, delay variation IPDV, and throughput). Results show that the Bi-LSTM controller reduces the standard deviation of CPU usage across servers from 43.39% to 4.18% under concurrent mixed traffic; contains CPU saturation at 75.49% compared to the 99.95% recorded by Round Robin in critical stress scenarios; sustains a packet loss rate of 0.0% where the static method reaches 1.43%; and maintains an average latency of 1.11 ms versus 1.20 ms in extended stability tests. It is concluded that the Bi-LSTM architecture significantly outperforms the Round Robin protocol in efficiency, load distribution fairness, fault resilience, and service continuity, validating its operational viability in SDN environments.

Keywords: SDN, load balancing, LSTM, Round Robin, recurrent neural networks, OpenFlow.

LISTA DE SIGLAS

ANN. Artificial Neural Network (Red Neuronal Artificial)

API. Application Programming Interface (Interfaz de Programación de Aplicaciones)

ARP. Address Resolution Protocol (Protocolo de Resolución de Direcciones)

Bi-LSTM. Bidirectional Long Short-Term Memory (Memoria a Largo y Corto Plazo Bidireccional)

CLI. Command Line Interface (Interfaz de Línea de Comandos)

CNN. Convolutional Neural Network (Red Neuronal Convolucional)

CPU. Central Processing Unit (Unidad Central de Procesamiento)

DNS. Domain Name System (Sistema de Nombres de Dominio)

EVE-NG. Emulated Virtual Environment – Next Generation

GNS3. Graphical Network Simulator 3

HTTP. HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto)

IA. Inteligencia Artificial

ICMP. Internet Control Message Protocol (Protocolo de Mensajes de Control de Internet)

IEEE. Institute of Electrical and Electronics Engineers

IoT. Internet of Things (Internet de las Cosas)

IP. Internet Protocol (Protocolo de Internet)

IPDV. IP Packet Delay Variation (Variación del Retardo de Paquetes IP)

IPLR. IP Packet Loss Ratio (Tasa de Pérdida de Paquetes IP)

IPTD. IP Packet Transfer Delay (Retardo de Transferencia de Paquetes IP)

ITU-T. International Telecommunication Union – Telecommunication Standardization Sector

JSON. JavaScript Object Notation (Notación de Objetos de JavaScript)

LSTM. Long Short-Term Memory (Memoria a Largo y Corto Plazo)

MAC. Media Access Control (Control de Acceso al Medio)

MLP. Multilayer Perceptron (Perceptrón Multicapa)

MPLS. Multiprotocol Label Switching (Conmutación de Etiquetas Multiprotocolo)

NAT. Network Address Translation (Traducción de Direcciones de Red)

NFV. Network Functions Virtualization (Virtualización de Funciones de Red)

NS-3. Network Simulator 3

ODS. Objetivos de Desarrollo Sostenible

ODL. OpenDaylight

ONOS. Open Network Operating System

POX. Python-based OpenFlow Controller

QoS. Quality of Service (Calidad de Servicio)

RAM. Random Access Memory (Memoria de Acceso Aleatorio)

RFC. Request for Comments

RNN. Recurrent Neural Network (Red Neuronal Recurrente)

RR. Round Robin

SDN. Software-Defined Networking (Redes Definidas por Software)

TCP. Transmission Control Protocol (Protocolo de Control de Transmisión)

UDP. User Datagram Protocol (Protocolo de Datagrama de Usuario)

URL. Uniform Resource Locator (Localizador de Recursos Uniforme)

VIP. Virtual IP (IP Virtual)

VoIP. Voice over IP (Voz sobre IP)

WRR. Weighted Round Robin

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS **11**

ÍNDICE DE TABLAS **13**

ÍNDICE DE FIGURAS **15**

INTRODUCCIÓN **16**

CAPITULO I ANTECEDENTES **17**

1.1 Problema de investigación. **17**

1.2 Objetivos. **19**
1.2.1 Objetivo General. 19
1.2.2 Objetivos Específicos. 19

1.3 Alcance. **20**

1.4 Justificación. **22**

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA **24**

2.1 Redes Definidas por Software (SDN) **25**
2.1.1 Arquitectura de redes SDN 26
2.1.2 Protocolo Openflow 30
2.1.2.1 Versiones del Protocolo Openflow 30

2.2 Balanceo de Carga en Redes SDN **34**
2.2.1 Técnicas tradicionales de balanceo de carga en redes SDN 35
2.2.1.1 El protocolo Round Robin 38
2.2.2 Técnicas de balanceo de carga con redes neuronales en redes SDN 42
2.2.2.1 Modelo Long Short-Term Memory 44

2.3 Herramientas de Software SDN **49**
2.3.1 Plataformas de simulación de redes 50
2.3.2 Controladores SDN 53

2.4 Modelo de Rendimiento UF1880 ..... **55**

CAPÍTULO III: DISEÑO E IMPLEMENTACIÓN **57**
3.1.1 Justificación del uso de la metodología en cascada 57
3.1.2 Fases del ciclo de vida aplicadas al proyecto 58

3.2 Análisis y Requisitos del Sistema **61**
3.2.1 Estudio de viabilidad 61
3.2.2 Requisitos funcionales y no funcionales 62
3.2.3 Requisitos de hardware y software 64

3.3 Diseño del sistema **67**

3.3.1 Arquitectura general SDN....67
3.3.2 Diseño de topología de red....68
3.3.3 Diseño del algoritmo Round Robin....69
3.3.4 Diseño del modelo neuronal (LSTM)....72
3.3.5 Creación de Dataset....75
3.3.6 Entrenamiento de LSTM....79

3.4 Implementación....85
3.4.1 Configuración del entorno experimental (Mininet, Ryu, OpenFlow 1.3)....86
3.4.2 Construcción de la topología simulada....88
3.4.3 Implementación de controlador Round Robin....92
3.4.4 Implementación de controlador RNN-LSTM....95

3.5 Técnicas de Análisis de Datos....99
3.5.1 Definición de métricas de rendimiento....99

CAPÍTULO IV: PRUEBAS.... 101

4.1 Definición de las Pruebas y Escenarios de Carga....101
4.1.1 Configuración de las ráfagas de tráfico y parámetros de simulación.....102
4.1.2 Procedimiento de recolección de telemetría....104
4.1.3 Diseño de los Casos de Prueba....106

4.2 Métricas de Evaluación del Desempeño....130
4.2.1 Latencia (Tiempo de Respuesta).....130
4.2.2 Rendimiento de ancho de Banda (Throughput)....131
4.2.3 Tasa de Pérdida de Paquetes (*Packet Loss*).....133
4.2.4 Equidad en la Distribución de Carga (Desviación Estándar de CPU/RAM).....134

4.3 Resultados y Análisis Comparativo (Round Robin vs. RNN-LSTM)....135
4.3.1 Desempeño bajo Tráfico Base y Operación Diaria *(Análisis de Pruebas 1 y 2).*....136
4.3.1.1 Análisis de la Prueba 1....136
4.3.1.2 Análisis de la Prueba 2....137
4.3.2 Comportamiento ante Saturación y Tráfico Pesado *(Análisis de Pruebas 3, 4 y 8).*....139
4.3.2.1 Análisis de la Prueba 3....139
4.3.2.2 Análisis de la Prueba 4....140
4.3.2.3 Análisis de la Prueba 8....141
4.3.3 Respuesta a Anomalías: Degradación, Congestión y Fallos *(Análisis de Pruebas 5, 6, 7 y 9).* 142
4.3.3.1 Análisis de la Prueba 5....142
4.3.3.2 Análisis de la Prueba 6....143
4.3.3.3 Análisis de la Prueba 7....144
4.3.3.4 Análisis de la Prueba 9....145
4.3.4 Análisis de Resiliencia y Estabilidad a Largo Plazo *(Análisis de Prueba 10).*....147

4.4 Evaluación Interna del Modelo Inteligente....148
4.4.1 Precisión de las decisiones de la IA en tiempo real.....148
4.4.2 Impacto computacional del controlador en el entorno SDN.....152

5 CONCLUSIONES Y RECOMENDACIONES.... 154

5.1 Conclusiones....154

5.2 Recomendaciones....158

6 REFERENCIAS BIBLIOGRÁFICAS 159

7 ANEXOS 165

7.1 Anexo A: Implementación de la Topología de Red (Mininet) 165

7.2 Anexo B. Lógica Base del Controlador Estático (Round Robin) 168

7.3 Anexo C. Lógica de Inferencia del Controlador Inteligente (LSTM) 172

7.4 Anexo D. Scripts de Automatización de Pruebas y Generación de Tráfico 175

7.5 Anexo E. Estructura de Telemetría y Registro de Decisiones (JSON) 178

7.6 Anexo F. Interfaz de Monitoreo y Dashboard de Control SDN 180

ÍNDICE DE TABLAS

Tabla 1. *Comparación del Rendimiento de algoritmos Tradicionales de balanceo de carga en SDN 37*
Tabla 2. *Rendimiento del algoritmo en balanceo de carga 41*
Tabla 3. *Comparación de modelos de redes neuronales 44*
Tabla 4. *Comparación de Simuladores de red 52*
Tabla 5. *Comparación de Controladores 54*
Tabla 6. *Requisitos funcionales 62*
Tabla 7. *Requisitos no funcionales 63*
Tabla 8. *Requerimientos de simuladores de red 64*
Tabla 9. *Requisitos del Controlador 65*
Tabla 10. *Requerimientos para correr Mininet 66*
Tabla 11. *Parámetros para procesamiento 74*
Tabla 12. *Librerías de Simulación de Red 87*
Tabla 13. *Librerías de Ciencia de Datos e Inteligencia Artificial 88*
Tabla 14. *Definición de variables del entorno de simulación. 88*
Tabla 15. *Variables y funciones principales del Controlador Round Robin 93*
Tabla 16. *Variables y funciones del sistema de balanceo inteligente LSTM 96*
Tabla 17. *Resumen de pruebas realizadas 106*
Tabla 18. *Variables y parametros de la prueba 1 ..... 110*
Tabla 19. *Variables y parámetros de la prueba 2 ..... 112*
Tabla 20. *Variables y parámetros de la prueba 3 ..... 114*
Tabla 21. *Variables y parámetros de la prueba 4 ..... 116*
Tabla 22. *Variable y parámetros de la prueba 5 ..... 118*
Tabla 23. *Variables y parámetros de la prueba 6 ..... 120*
Tabla 24. *Variables y parámetros de la prueba 7 ..... 122*
Tabla 25. *Variables y parámetros de la prueba 8 ..... 124*
Tabla 26. *Variables y parámetros de la prueba 9 ..... 126*
Tabla 27. *Variables y parámetros de la prueba 10 ..... 128*
Tabla 28. *Resultados comparativos de la Prueba 1 ..... 137*
Tabla 29. *Resultados comparativos de la Prueba 2 ..... 138*
Tabla 30. *Resultados comparativos de la Prueba 3 (Saturación) 139*
Tabla 31. *Resultados comparativos de la Prueba 4 (Streaming) 140*
Tabla 32. *Resultados comparativos de la Prueba 8 (Tráfico Mixto) 141*

Tabla 33. *Resultados comparativos de la Prueba 5 (Ráfagas Asimétricas)* *143*
Tabla 34. *Resultados comparativos de la Prueba 6 (Degradación por Jitter)* *144*
Tabla 35. *Resultados comparativos de la Prueba 7 (Congestión en el Núcleo)* *145*
Tabla 36. *Resultados comparativos de la Prueba 9 (Caída de Nodo)* *146*
Tabla 37. *Resultados comparativos de la Prueba 10 (Estabilidad a Largo Plazo)* *147*
Tabla 38. *Divergencia de enrutamiento (LSTM frente a Round Robin)* *150*
Tabla 39. *Eventos de conmutación proactiva por saturación de CPU* *151*
Tabla 40*. Impacto en Latencia (IPTD) y Volumen de Decisiones Lógicas* *152*

ÍNDICE DE FIGURAS

Figura 1. *Arquitectura genérica de la propuesta del proyecto.* ____ 21
Figura 2. *Arquitectura de una red SDN* ____ 27
Figura 3. *Elementos de una arquitectura SDN* ____ 29
Figura 4. *Versiones de Openflow* ____ 31
Figura 5. *Funcionamiento de Round Robin* ____ 39
Figura 6. *Aplicación del algoritmo Round Robin en el balanceo de carga* ____ 40
Figura 7. *Representación del funcionamiento de interno de LSTM* ____ 47
Figura 8*. Fases del Proyecto* ____ 60
Figura 9. *Topología de Red experimental SDN* ____ 69
Figura 10. *Diagrama de Bloques de LSTM* ____ 74
Figura 11. *Lógica de preprocesamiento, normalización y transformación a tensores.* ____ 78
Figura 12*. Curvas de precisión y pérdida (Entropía Cruzada) durante el entrenamiento del modelo LSTM.* ____ 83
Figura 13. *Matriz de Confusión evaluando la exactitud predictiva del modelo sobre el clúster de 4 servidores.* ____ 84
Figura 14. *Ejecución lógica del flujo de entrenamiento y validación de la red neuronal LSTM.* ____ 85
Figura 15. *Algoritmo de despliegue de infraestructura SDN* ____ 91
Figura 16. *Algoritmo de balanceo de carga Round Robin con gestión de identidades virtuales y persistencia de sesión.* ____ 94
Figura 17. *Algoritmo de balanceo con RNN-LSTM* ____ 97
Figura 18. *Flujo de ejecución de pruebas* ____ 109
Figura 19. *Algoritmo de la Prueba 1 (Tráfico Base)* ____ 111
Figura 20*. Algoritmo de la Prueba 2 (Operación Diaria)* ____ 113
Figura 21. *Algoritmo de la Prueba 3 (Estrés Crítico)* ____ 115
Figura 22. *Algoritmo de la Prueba 4 (Tráfico Pesado - Streaming)* ____ 117
Figura 23. *Algoritmo de la Prueba 5 (Ráfagas Súbitas - Spikes)* ____ 119
Figura 24. *Algoritmo de la Prueba 6 (Degradación - Jitter)* ____ 121
Figura 25. *Algoritmo de la Prueba 7 (Congestión Core)* ____ 123
Figura 26. *Algoritmo de la Prueba 8 (Tráfico Mixto)* ____ 125
Figura 27. *Algoritmo de la Prueba 9 (Fallo de Nodo - Failover)* ____ 127
Figura 28. *Algoritmo de la Prueba 10 (Estabilidad a Largo Plazo)* ____ 129

INTRODUCCIÓN

La gestión del tráfico de datos en las arquitecturas de Redes Definidas por Software (SDN) enfrenta el desafío de soportar aplicaciones con alta variabilidad , donde el uso de algoritmos estáticos tradicionales como Round Robin genera saturación asimétrica en los servidores y una degradación directa de la Calidad de Servicio (QoS). Para superar estas limitaciones operativas, la presente investigación desarrolla un análisis comparativo entre dicho protocolo secuencial y una técnica predictiva basada en Redes Neuronales Recurrentes (RNN), específicamente el modelo Long Short-Term Memory (LSTM), integrado de forma nativa en un controlador SDN. Mediante una metodología en cascada sobre un entorno controlado de simulación , el estudio evalúa métricas críticas de rendimiento que incluyen la latencia, la pérdida de paquetes, el uso de ancho de banda y la utilización de recursos computacionales bajo distintos niveles de carga y estrés. El propósito central radica en cuantificar la eficiencia y adaptabilidad de la inteligencia artificial para aprender patrones temporales complejos y anticipar la demanda de la red , demostrando empíricamente cómo la distribución dinámica del tráfico optimiza la infraestructura y previene los cuellos de botella inherentes a los métodos convencionales.

CAPITULO I ANTECEDENTES

En este capítulo presenta el planteamiento del problema, que constituye la base de este proyecto, así como los objetivos que se buscan alcanzar. A continuación, se define el alcance de la investigación, estableciendo los límites de este estudio. Finalmente, se expone la justificación que respalda la realización del proyecto, proporcionando una comprensión detallada de la importancia y la necesidad de llevar a cabo este trabajo.

1.1 **Problema de investigación.**

El balanceo de carga es un ámbito importante en la gestión de redes, especialmente en aquellas con un alto volumen de tráfico y múltiples servicios que requieren una distribución eficiente de las solicitudes entre los servidores disponibles. Las redes tradicionales, que utilizan algoritmos estáticos como Round Robin, han sido ampliamente empleados en el pasado debido a su simplicidad y eficacia en redes pequeñas o con tráfico predecible. Sin embargo, a medida que las redes evolucionan y aumentan en complejidad, la demanda por una mayor flexibilidad y adaptabilidad ha puesto en evidencia las limitaciones de estos algoritmos.

Los algoritmos tradicionales como Round Robin distribuyen las solicitudes de manera cíclica entre los servidores sin tener en cuenta las variaciones en el rendimiento o la carga de los servidores. Este enfoque, aunque efectivo en situaciones simples, no es capaz de adaptarse dinámicamente a las fluctuaciones del tráfico en tiempo real, lo que puede llevar a la saturación de ciertos servidores mientras otros permanecen poco usados. En redes de gran escala, como las implementadas en redes definidas por software (SDN), donde las condiciones cambian constantemente debido a la alta demanda, estos

algoritmos estáticos ya no son suficientes para garantizar un rendimiento óptimo (Chahlaoui et al., 2019).

El tráfico de datos moderno, impulsado por aplicaciones que requieren grandes anchos de banda, como la transmisión en tiempo real o los servicios basados en la nube, introduce una variabilidad mucho mayor en la carga de la red, lo que demanda soluciones de balanceo de carga que sean más dinámicas y adaptativas. En este contexto, las redes tradicionales, que operan bajo principios estáticos y predeterminados, se enfrentan a la dificultad de gestionar redes con requisitos de alto rendimiento y disponibilidad, lo que genera un mayor riesgo de congestión y reducción en la calidad del servicio (Baniya et al., 2024).

Además, el gran crecimiento del tráfico y la implementación de redes más complejas requieren que los sistemas sean capaces de predecir y adaptarse a las necesidades cambiantes de la red. Las redes definidas por software (SDN), con su capacidad para centralizar el control y tomar decisiones basadas en una visión global de la red, brindan un nuevo paradigma para abordar estos desafíos. Sin embargo, para aprovechar completamente las ventajas de SDN, es necesario desarrollar soluciones de balanceo de carga que no dependan de los algoritmos estáticos tradicionales, sino que estén diseñadas para adaptarse a las demandas dinámicas de tráfico (Gures et al., 2022).

La necesidad de superar las limitaciones de los algoritmos de balanceo de carga tradicionales ha llevado a la exploración de métodos más sofisticados que son capaces de ajustarse en tiempo real a las condiciones cambiantes de la red. Este cambio es importante para garantizar la eficiencia operativa y el rendimiento en redes de gran escala y de alta disponibilidad (Omer et al., 2021).Sin un enfoque adecuado para la distribución del tráfico, las redes modernas corren el riesgo de sufrir problemas como la congestión y la disminución en la calidad de los servicios ofrecidos, lo que impacta directamente en la

experiencia del usuario y en la eficiencia operativa de las empresas que dependen de estos sistemas.

Adicionalmente en la Universidad Técnica del Norte, se evidencia una falta de investigación orientada a comprender, de manera práctica, el funcionamiento del balanceo de carga en redes definidas por software (SDN). Esta carencia no solo limita la generación de conocimiento local en torno a tecnologías emergentes, sino que también restringe el desarrollo de soluciones innovadoras que puedan responder a las demandas de las redes modernas. La ausencia de estudios que exploren enfoques dinámicos y adaptativos para la gestión del tráfico obstaculiza el fortalecimiento de la capacidad investigativa de la universidad, reduciendo su competitividad académica. Es necesario promover trabajos que no solo profundicen en las bases teóricas del balanceo de carga, sino que también presenten aplicaciones prácticas que impacten positivamente el entorno académico y profesional.

1.2 **Objetivos.**

1.2.1 Objetivo General.

Analizar el desempeño del balanceo de carga basado en redes neuronales para redes SDN en un entorno controlado de simulación

1.2.2 Objetivos Específicos.

1. Revisar el estado del arte sobre técnicas de balanceo de carga en redes definidas por software (SDN), con énfasis en el protocolo Round Robin y las redes neuronales recurrentes (RNN) específicamente en el modelo Long Short-Term Memory (LSTM), para fundamentar teóricamente la investigación.

2. Implementar los algoritmos Round Robin y RNN siguiendo las etapas de la metodología en cascada, asegurando su integración funcional dentro del entorno SDN.
3. Medir el rendimiento de cada método utilizando métricas como latencia, pérdida de paquetes, uso de ancho de banda y utilización de recursos, en escenarios con distintos niveles de carga de red.
4. Identificar el nivel de eficiencia y adaptabilidad de cada técnica en la gestión del tráfico dentro de redes SDN en función de los resultados obtenidos en las simulaciones.

1.3 **Alcance.**

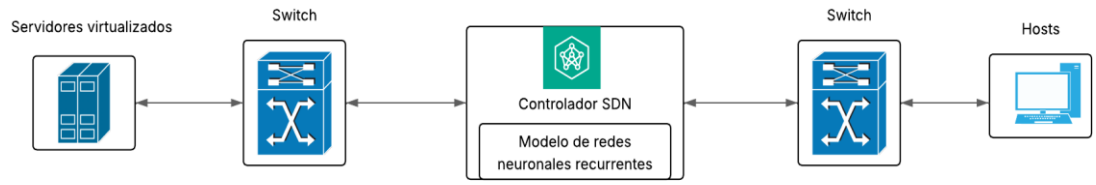
El presente proyecto pretende realizar un estudio comparativo entre dos enfoques de balanceo de carga en redes definidas por software (SDN), por un lado, a través del protocolo estático Round Robin y por otro las redes neuronales recurrentes (RNN). El desarrollo del proyecto seguirá la metodología en cascada, permitiendo una secuencia clara y estructurada de las fases del proyecto. Este enfoque metodológico asegura que cada etapa se complete de manera secuencial y lógica, permitiendo que los resultados obtenidos en una fase sirvan como base para el desarrollo de la siguiente.

En la primera fase, se llevará a cabo una revisión teórica de las técnicas de balanceo de carga utilizadas en redes SDN, el protocolo Round Robin y las redes neuronales recurrentes (RNN), específicamente el modelo Long Short-Term Memory (LSTM). Esta revisión incluirá la exploración de literatura científica, artículos académicos, libros y documentos técnicos que describan las características fundamentales de cada técnica. El análisis abarcará aspectos como su historia, evolución, aplicaciones y su impacto en el rendimiento de las redes SDN. Además, se analizarán estudios previos

que hayan comparado métodos de balanceo de carga similares, identificando las principales conclusiones y vacíos de conocimiento que este proyecto buscará abordar. Esta revisión permitirá establecer las bases teóricas necesarias para la correcta implementación y evaluación de ambos enfoques en el entorno SDN.

Durante la segunda fase, se procederá a la implementación del protocolo Round Robin y del modelo de redes neuronales recurrentes (RNN) como técnicas de balanceo de carga. Inicialmente, se llevará a cabo la configuración del entorno SDN, asegurando que todos los componentes necesarios, tales como el controlador SDN, los switches y los servidores virtualizados, estén correctamente interconectados según la arquitectura descrita en la Figura 2. El protocolo Round Robin será implementado directamente en el controlador SDN, configurado para distribuir el tráfico de manera equitativa entre los recursos disponibles. Por otro lado, el modelo de RNN estará basado en la arquitectura Long Short-Term Memory (LSTM), conocida por su capacidad para aprender patrones temporales complejos. Este modelo será entrenado utilizando datos de tráfico simulado, permitiendo que aprenda patrones de comportamiento en la red y optimice la asignación de recursos. Este modelo se integrará al controlador SDN, proporcionando una solución dinámica y adaptativa de balanceo de carga.

Figura 1. *Arquitectura genérica de la propuesta del proyecto.*



Una vez implementados los algoritmos, se procederá a medir su rendimiento mediante pruebas y simulaciones. Las métricas de evaluación incluirán la latencia, que se refiere al tiempo de respuesta del sistema para cada solicitud; la pérdida de paquetes, que

indicará la cantidad de información que no llega a su destino; el uso de ancho de banda, que reflejará la eficiencia en el uso de los recursos de red; y la utilización de recursos del sistema, que abarcará el consumo de CPU y memoria del controlador SDN. Las pruebas se llevarán a cabo en diferentes escenarios de carga, variando la cantidad de solicitudes y la intensidad del tráfico, lo cual permitirá evaluar el desempeño de cada técnica bajo condiciones controladas.

En la última fase, se analizarán los resultados obtenidos para comparar la eficiencia de los dos enfoques implementados. Los datos recopilados serán procesados y presentados en forma de tablas y gráficos que faciliten su interpretación. El análisis permitirá identificar las fortalezas y debilidades de cada técnica de balanceo de carga, explicando las causas de las diferencias observadas en el rendimiento. Además, se discutirán posibles mejoras para cada método y se ofrecerán recomendaciones para su implementación en entornos SDN reales. Finalmente, se establecerán conclusiones claras que reflejen los resultados del estudio

1.4 **Justificación.**

El presente proyecto se justifica desde diversas perspectivas: tecnológica, teórica y metodológica, alineándose a los Objetivos de Desarrollo Sostenible (ODS), específicamente en los objetivos relacionados con la innovación tecnológica (ODS 9: Industria, Innovación e Infraestructura), el acceso a la energía asequible y no contaminante (ODS 7), y la reducción de costos operativos mediante tecnologías de computación en la nube (ODS 12: Producción y Consumo Responsables).

Desde una perspectiva tecnológica, este proyecto se centra en el uso de redes neuronales recurrentes (RNN), específicamente el modelo Long Short-Term Memory (LSTM), como solución avanzada para el balanceo de carga en redes definidas por

software (SDN). Las RNN-LSTM son particularmente eficaces para capturar y predecir patrones temporales complejos en el tráfico de red, lo que les permite adaptarse dinámicamente a las condiciones cambiantes de la red. A diferencia de los métodos tradicionales, como los algoritmos estáticos, las LSTM utilizan su capacidad de memoria para analizar secuencias de datos, identificando patrones que optimizan la distribución del tráfico y mejoran el uso de los recursos de red. Según (Mohammed et al., 2019), las RNN-LSTM no solo mejoran la eficiencia del balanceo de carga, sino que también reducen la latencia y minimizan la pérdida de paquetes, proporcionando una solución adaptativa y robusta para redes SDN.

Además, este proyecto promueve la implementación de soluciones basadas en la alta demanda de aplicaciones críticas como telefonía IP, bases de datos y streaming, lo que facilita implementar una infraestructura avanzada sin necesidad de grandes inversiones iniciales. Esto contribuye a reducir costos operativos y de mantenimiento, haciéndolo una opción accesible para pequeñas y medianas empresas que buscan optimizar sus recursos tecnológicos (Gures et al., 2022). Adicionalmente, (Tayfour et al., 2023)destacan que las técnicas basadas en inteligencia artificial, como las RNN-LSTM, pueden mejorar la eficiencia del balanceo de carga en entornos de red SDN, al adaptarse dinámicamente a las condiciones cambiantes del tráfico.

Teóricamente, esta investigación amplía el conocimiento existente en el campo de las redes definidas por software (SDN) y las técnicas de balanceo de carga, aportando un enfoque comparativo que permite evaluar la eficiencia del protocolo Round Robin frente al modelo LSTM de redes neuronales recurrentes. Este análisis contribuye al debate académico sobre las mejores estrategias para la gestión de tráfico en redes avanzadas, proporcionando datos experimentales que respaldan la superioridad de una u otra técnica bajo distintos escenarios de carga. Según (Mahmoudi et al., 2024), los modelos LSTM

pueden predecir patrones de tráfico de manera más precisa que los métodos tradicionales, lo que los hace adecuados para entornos SDN donde las condiciones de red cambian rápidamente.

Metodológicamente, este proyecto se justifica al proponer una metodología en cascada para la implementación de los algoritmos de balanceo de carga, permitiendo una secuencia clara y controlada de las etapas del proyecto. Esta metodología garantiza que cada fase se ejecute de manera ordenada, desde la revisión del estado del arte hasta el análisis comparativo de los resultados. Adicionalmente, la integración del modelo LSTM como parte del controlador SDN introduce una innovación metodológica al aplicar técnicas de inteligencia artificial en un entorno de red definido por software. Según (TONKAL & POLAT, 2021)el uso de modelos LSTM en la gestión de tráfico de red proporciona una precisión significativamente mayor en comparación con los métodos convencionales

2 **CAPÍTULO II:** FUNDAMENTACIÓN TEÓRICA

El presente capítulo establece el sustento teórico y conceptual indispensable para fundamentar el desarrollo, la implementación y la evaluación de la arquitectura propuesta. En primer lugar, se examinan los principios de las Redes Definidas por Software (SDN) y la desagregación nativa entre el plano de control y el plano de datos bajo el estándar del protocolo OpenFlow. Posteriormente, se analizan los mecanismos tradicionales de distribución de tráfico, como el algoritmo Round Robin, identificando sus limitaciones operativas frente a la concurrencia de flujos dinámicos multiprotocolo. Asimismo, se introducen los fundamentos de la Inteligencia Artificial y el aprendizaje profundo, justificando formalmente la selección de la arquitectura de Memoria a Largo y Corto

Plazo Bidireccional (Bi-LSTM) para el modelado predictivo de las series temporales de telemetría. Finalmente, se describen las herramientas de software especializadas en la emulación y simulación de redes, estructurando de este modo el soporte científico necesario para comprender la metodología experimental y el posterior análisis comparativo de los resultados.

2.1 Redes Definidas por Software (SDN)

Las Redes Definidas por Software constituyen un modelo arquitectónico avanzado que redefine la gestión y operación de las redes al desacoplar el plano de control del plano de datos, eliminando la dependencia jerárquica entre ambos y centralizando las decisiones de enrutamiento en un controlador lógico externo (Barrera Pérez & Serrato Losada, 2019; Rodríguez Herlein et al., 2019). Este modelo introduce un plano de control programable que actúa como núcleo de inteligencia de la red, mientras que los dispositivos de conmutación y enrutamiento se limitan a ejecutar las instrucciones recibidas, optimizando así el flujo de tráfico y el uso de recursos. Según Haji et al. (2021), esta separación funcional transforma la infraestructura tradicional en un entorno altamente dinámico, en el que la administración y la orquestación de políticas se realizan mediante interfaces abiertas (APIs), lo que permite automatizar tareas complejas, desplegar servicios bajo demanda y adaptar el comportamiento de la red a condiciones cambiantes en tiempo real. Entre las propiedades técnicas más relevantes se incluyen el control centralizado, que proporciona una visión global y coherente de la topología; la programabilidad, que habilita la integración con aplicaciones de gestión inteligente; la flexibilidad operacional, que permite reconfigurar la red de manera reactiva ante fallos o congestión; y la independencia del hardware, que promueve la interoperabilidad entre equipos heterogéneos. En conjunto, SDN se erige como una solución a la rigidez y

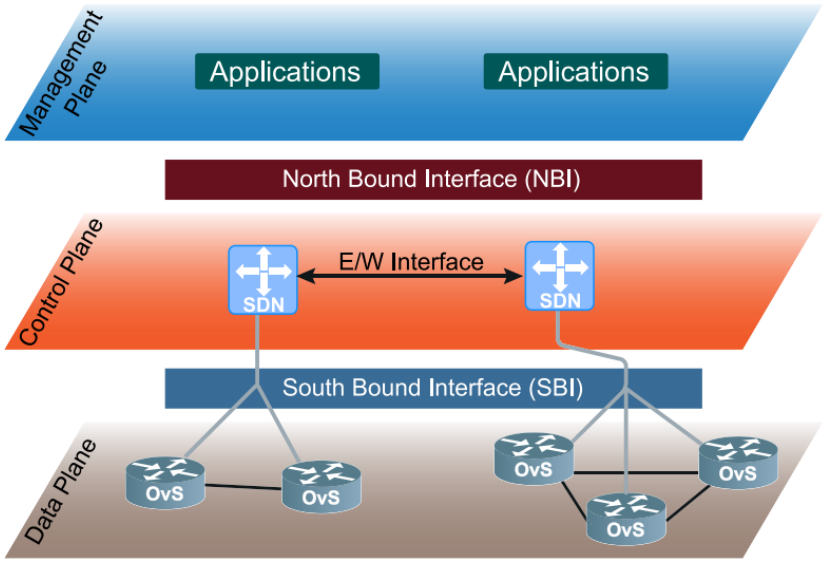
fragmentación de las redes convencionales, al ofrecer un marco arquitectónico unificado que facilita la virtualización, la automatización y la optimización integral del tráfico en entornos de comunicaciones modernas.

2.1.1 Arquitectura de redes SDN

La arquitectura de SDN está estructurada en tres capas fundamentales que separan la lógica de control de los dispositivos de red físicos. Esta organización funcional puede observarse en la Figura 2, donde se representa gráficamente la interacción entre las capas de aplicación, control y datos, así como las interfaces que permiten su comunicación y cooperación efectiva.

- **Plano de datos (Data Plane):** compuesto por dispositivos físicos o virtuales (switches, routers, access points) que reenvían paquetes bajo instrucciones del controlador. Estos dispositivos actúan como elementos de reenvío, sin capacidad de decisión autónoma.
- **Plano de control (Control Plane):** contiene el controlador SDN que centraliza la información de red y define las políticas de reenvío. Interactúa con los dispositivos mediante protocolos como OpenFlow, proporcionando una visión abstracta del estado de la red y posibilitando una gestión optimizada de los recursos.
- **Plano de aplicación (Application Plane):** capa superior donde residen aplicaciones que definen políticas de red específicas (como seguridad, enrutamiento, balanceo de carga, análisis de tráfico, etc.), comunicándose con el controlador mediante APIs Northbound (Haji et al., 2021; Rodríguez Herlein et al., 2019). Estas aplicaciones traducen los requerimientos de negocio en reglas concretas que serán implementadas por el controlador.

Figura 2. *Arquitectura de una red SDN*



Fuente: (Latif et al., 2020).

Adicionalmente, la arquitectura SDN que se muestra en la figura 2 se logra contemplar las siguientes interfaces:

- **Southbound API:** conecta el plano de control con el de datos. El protocolo más común es OpenFlow, aunque existen otros como NETCONF y OVSDB. Estas APIs permiten al controlador instalar reglas, recopilar estadísticas y consultar el estado de los dispositivos de red.
- **Northbound API:** permite la comunicación entre las aplicaciones de usuario y el controlador. Aunque no existe una estandarización formal, se utilizan comúnmente APIs RESTful, que facilitan la integración con plataformas de gestión y orquestación.

Esta arquitectura jerárquica de SDN aporta mayor claridad en el diseño y operación de redes, sino que también posibilita una evolución más rápida de las tecnologías de red, al independizar el desarrollo de servicios de las restricciones impuestas por el hardware.

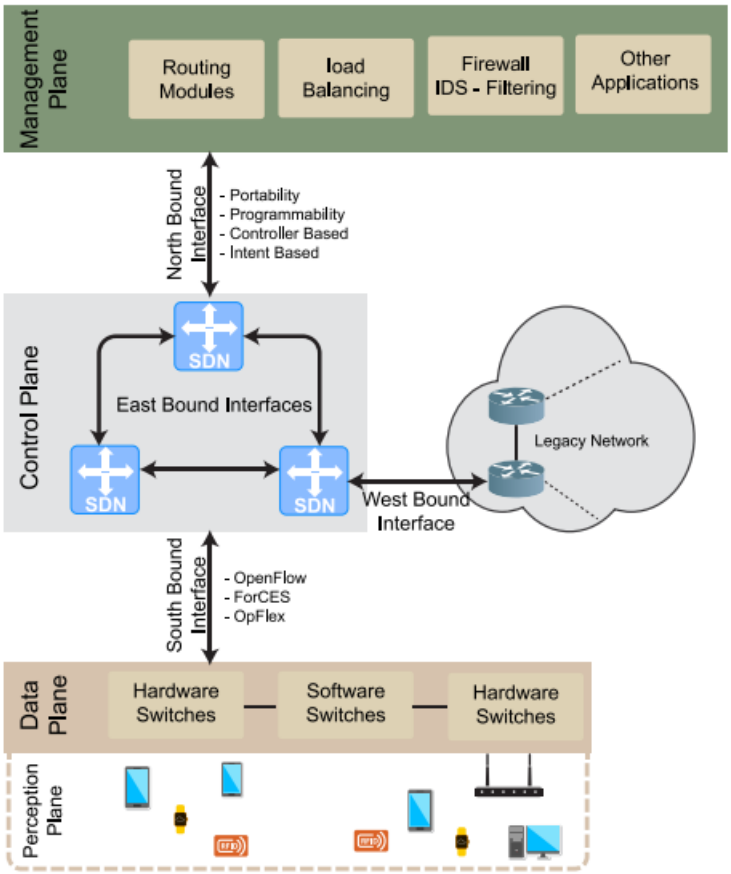
A nivel de componentes, los elementos clave de una arquitectura SDN se pueden comprender de forma más clara con base en la Figura 3, la cual ilustra de manera detallada cómo interactúan los diferentes planos y módulos dentro de un entorno SDN. Esta figura representa la integración entre los planos de gestión, control y datos, y muestra explícitamente la relación entre los controladores, las interfaces y los dispositivos de red.

- **Controlador SDN:** Como se muestra en el plano de control de la Figura 3, el controlador actúa como el cerebro de la red. Es el encargado de recopilar el estado de los dispositivos de red (tanto físicos como virtuales), procesar la información recibida desde los switches y routers, y tomar decisiones de enrutamiento o aplicación de políticas. Su diseño puede ser centralizado o distribuido, como lo muestra la conexión entre múltiples controladores mediante interfaces Eastbound, lo cual contribuye a la escalabilidad y redundancia del sistema (Latif et al., 2020)
- **Dispositivos de red programables:** en el plano de datos (Data Plane), la Figura 3 muestra la coexistencia de dispositivos físicos (Hardware Switches) y virtuales (Software Switches), encargados del reenvío de paquetes. Estos dispositivos siguen las instrucciones dictadas por el controlador a través de protocolos como OpenFlow, ForCES o OpFlex, todos ellos representados como parte de la interfaz Southbound. La capacidad de programación de estos dispositivos permite implementar reglas de reenvío dinámicamente y responder en tiempo real a los cambios en la red.
- **Aplicaciones SDN:** situadas en el plano de gestión (Management Plane), estas aplicaciones incluyen módulos de enrutamiento, balanceo de carga, firewall, IDS/IPS, entre otros. Estas funcionalidades se desarrollan sobre el controlador utilizando APIs Northbound. Como se evidencia en la Figura 3, estas aplicaciones se comunican directamente con el controlador SDN, definiendo comportamientos

y políticas que luego se traducen en reglas de reenvío implementadas en el plano de datos.

- **Interfaces y protocolos:** la arquitectura SDN se apoya en un conjunto de interfaces que permiten la interacción entre planos. La Northbound Interface (NBI) proporciona la conexión entre el controlador y las aplicaciones, mientras que la Southbound Interface (SBI) gestiona la comunicación con los dispositivos de red. Adicionalmente, se contemplan las Eastbound y Westbound Interfaces, que permiten la interconexión entre múltiples controladores o con redes legadas, tal como se representa en la conexión con una "Legacy Network" en la Figura 3.

Figura 3. *Elementos de una arquitectura SDN*



Fuente: (Latif et al., 2020)

Estas características y componentes permiten que las Redes Definidas por Software ofrezcan una serie de ventajas.

2.1.2 Protocolo Openflow

OpenFlow es el protocolo principal que posibilita la comunicación directa entre el plano de control y el plano de datos dentro del paradigma de las Redes Definidas por Software (SDN). Este protocolo establece una interfaz estandarizada que permite al controlador SDN gestionar de manera remota el comportamiento de los dispositivos de red (switches y routers), sin requerir modificaciones en el hardware subyacente (Toledo Gómez, 2023). A diferencia de las redes tradicionales, donde las decisiones de enrutamiento y el reenvío de paquetes se encuentran integrados en un mismo dispositivo, OpenFlow permite que dichas decisiones sean tomadas de forma centralizada por el controlador, mientras los switches se limitan a ejecutar las reglas de reenvío previamente definidas.

El protocolo OpenFlow emplea un canal de control seguro (OpenFlow Channel) entre el controlador y los switches, sobre el cual se intercambian mensajes que configuran, actualizan y consultan las tablas de flujo. Estos mensajes pueden clasificarse en tres tipos principales: mensajes de control (controller-to-switch), mensajes asincrónicos (switch-to-controller) y mensajes simétricos, cada uno con funciones específicas en la gestión del tráfico y la supervisión del estado de la red (Shi et al., 2020)

2.1.2.1 Versiones del Protocolo Openflow

Desde su lanzamiento en 2009, el protocolo OpenFlow ha evolucionado de manera progresiva para adaptarse a las crecientes necesidades de control, flexibilidad y escalabilidad dentro de las redes definidas por software (SDN). Cada versión ha

incorporado mejoras orientadas a optimizar la gestión del tráfico, la compatibilidad con nuevos protocolos y la capacidad de administración remota. Estas actualizaciones han permitido una mayor eficiencia en la implementación de políticas de enrutamiento, seguridad y balanceo de carga, respondiendo a las redes modernas caracterizadas por la virtualización y el incremento del tráfico de datos (Shi et al., 2020; Costa et al., 2021). La Figura 4 muestra una síntesis de las principales versiones del protocolo OpenFlow y las funciones que caracterizan a cada una de ellas (Wang et al., 2018).

Figura 4. *Versiones de Openflow*



Fuente: (Latif et al., 2020)

OpenFlow 1.0 (2009):

Fue la versión inicial del protocolo, introduciendo el concepto de tabla de flujos única (Single Table) con campos de coincidencia fijos (Fixed Matching Fields). Su diseño estaba enfocado en la experimentación y en validar la comunicación entre el controlador SDN y los dispositivos de red, estableciendo la base del modelo de separación entre el plano de control y el plano de datos (Gómez, 2012)

OpenFlow 1.1 (2011):

Esta versión incorporó soporte para múltiples tablas de flujo (Multiple Tables) y tablas de grupo (Group Tables), las cuales permitieron un control más granular y flexible del tráfico. Además, añadió compatibilidad con VLAN y MPLS, facilitando la aplicación de OpenFlow en entornos de operadores y proveedores de servicios. Con estas mejoras,

el protocolo comenzó a soportar balanceo de carga básico y mecanismos de redundancia para mejorar la continuidad del servicio (Latif et al., 2020).

OpenFlow 1.2 (2011):

Introdujo la estructura OXM (OpenFlow Extensible Match) basada en el formato TLV (Type-Length-Value), que amplió la capacidad de coincidencia de campos de cabecera y permitió mayor personalización en las reglas de flujo. También se añadió soporte nativo para IPv6 y la funcionalidad de intercambio de roles entre controladores (Controller Role Exchange), mejorando la tolerancia a fallos y la gestión jerárquica de redes SDN (Shi et al., 2020).

OpenFlow 1.3 (2012):

Esta versión consolidó el uso del protocolo al introducir las tablas de medidor (Meter Tables), que permiten aplicar políticas de Quality of Service (QoS) mediante el control de tasas de tráfico. Asimismo, se incorporó la entrada table-miss, que define el comportamiento de los paquetes que no coinciden con ninguna regla configurada. Gracias a su estabilidad y soporte ampliado, OpenFlow 1.3 se convirtió en la versión más implementada en controladores como Ryu, POX y OpenDaylight (Lasso Guamán & Puchaicela Condoy, 2021; Shi et al., 2020).

OpenFlow 1.4 (2013):

Incluyó mejoras enfocadas en la sincronización de tablas (Synchronized Tables) y la función Bundle, la cual permite aplicar múltiples modificaciones en la tabla de flujos

como una sola transacción atómica. Esto redujo el riesgo de inconsistencias durante las actualizaciones y aumentó la confiabilidad del proceso de reconfiguración de red, especialmente en entornos distribuidos (Costa et al., 2021).

OpenFlow 1.5 (2015):

Representó una versión avanzada del protocolo, integrando las tablas de salida (Egress Tables), que permiten definir acciones sobre los paquetes justo antes de ser enviados a los puertos físicos, y los bundles programados (Scheduled Bundles), que posibilitan la ejecución de configuraciones temporizadas. Estas capacidades ofrecieron mayor control granular y soporte para automatización de tareas críticas, consolidando el protocolo como una herramienta adaptable a escenarios con alto dinamismo de tráfico y aplicaciones inteligentes basadas en predicción de carga (Latif et al., 2020; Shi et al., 2020).

- Estructura de trama OpenFlow

A nivel de estructura, una trama OpenFlow (OpenFlow Packet) está compuesta por campos bien definidos que permiten la identificación, manipulación y control del tráfico dentro del switch. En términos generales, cada mensaje OpenFlow posee los siguientes componentes:

- **Header (Cabecera):** contiene el campo *version*, que indica la versión del protocolo utilizada; *type*, que especifica el tipo de mensaje (por ejemplo, OFPT_FLOW_MOD para modificar una regla); *length*, que define el tamaño total del mensaje; y *transaction ID (xid)*, que permite asociar respuestas a solicitudes específicas.

- **Body (Cuerpo):** su estructura depende del tipo de mensaje. En el caso de un mensaje FLOW_MOD, el cuerpo contiene las **reglas de coincidencia (match fields)**, las **acciones a ejecutar (actions)** y los **parámetros de prioridad y tiempo de expiración**.
- **Footer (Opcional):** puede incluir datos adicionales, como estadísticas de flujo o identificadores de puerto, especialmente en mensajes de tipo *stats reply*.

El funcionamiento de estas tramas permite que el controlador SDN defina, en tiempo real, políticas de enrutamiento adaptativas. A través de los mensajes PACKET_IN, FLOW_MOD y STATS_REPLY, el controlador recopila información sobre el tráfico, detecta congestión y modifica las reglas de reenvío para redistribuir los flujos, favoreciendo así mecanismos avanzados de balanceo de carga dinámico (Gómez, 2012).

2.2 Balanceo de Carga en Redes SDN

El balanceo de carga es una técnica fundamental en el diseño de redes modernas, particularmente en infraestructuras centradas en centros de datos y servicios web. En entornos tradicionales, esta tarea se realiza mediante dispositivos físicos dedicados o mediante métodos como DNS Round Robin, los cuales presentan limitaciones significativas en términos de escalabilidad, adaptabilidad y confiabilidad. Con la aparición de Software Defined Networking (SDN), estas limitaciones han sido superadas mediante la centralización del control y la programabilidad de los dispositivos de red, lo que permite implementar algoritmos de balanceo de carga de forma flexible, eficiente y dinámica (Singh et al., 2022)

2.2.1 Técnicas tradicionales de balanceo de carga en redes SDN

Las técnicas tradicionales de balanceo de carga emplean algoritmos estáticos como Round Robin o Least Connections para distribuir el tráfico entre varios nodos de red. Estos métodos son simples y eficientes, pero carecen de adaptabilidad ante variaciones dinámicas del tráfico, lo que limita su rendimiento en entornos de alta demanda(Belgaum et al., 2020; Vishwakarma, 2022).

La Tabla 1 presenta un resumen comparativo de los resultados experimentales obtenidos al evaluar distintas técnicas tradicionales de balanceo de carga en redes SDN. Esta figura permite observar de forma cuantitativa el comportamiento de cada algoritmo en términos de tiempo de respuesta, tasa de transacciones, rendimiento (throughput) y concurrencia, sirviendo como base empírica para el análisis posterior.

Entre las técnicas tradicionales más destacadas aplicadas en entornos SDN, se encuentran:

- **Round Robin (RR):** Es una de las técnicas más simples y ampliamente utilizadas en entornos homogéneos. Su funcionamiento consiste en asignar las solicitudes entrantes a los servidores disponibles en orden secuencial y circular. Por ejemplo, si existen tres servidores activos, la primera solicitud será dirigida al servidor 1, la segunda al servidor 2, la tercera al servidor 3, y la cuarta de nuevo al servidor 1, repitiendo el ciclo. Esta técnica es efectiva cuando los servidores tienen capacidades similares, pero puede resultar ineficiente en entornos heterogéneos donde los recursos difieren (Vishwakarma, 2022).
- **Least Connection:** Esta técnica asigna la nueva solicitud al servidor que tenga la menor cantidad de conexiones activas en ese momento. Es especialmente útil en

entornos donde las sesiones varían en duración, intensidad o consumo de recursos. A diferencia del Round Robin, que no considera el estado actual de los servidores, Least Connection adapta la asignación de tráfico en tiempo real, evitando la sobrecarga de ciertos nodos. (Zhu et al., 2019a) señalan que esta técnica logra equilibrar la carga de manera eficiente en redes SDN, distribuyendo el tráfico según la disponibilidad real de los servidores.

- **Weighted Round Robin (WRR):** Es una extensión del algoritmo Round Robin que introduce un sistema de ponderación asignando a cada servidor un peso proporcional a su capacidad de procesamiento. De este modo, los servidores con mayores recursos gestionan un volumen superior de solicitudes, optimizando la distribución de carga en entornos heterogéneos. Tal como se muestra en la Tabla 1, WRR obtiene el mejor desempeño con un tiempo de respuesta promedio de 0.13 segundos y una tasa de transacciones de 23.91 por segundo, superando al Round Robin tradicional, que registra 0.16 segundos y 21.94 transacciones por segundo. En contraste, los algoritmos Equal Load, Least Connection y Random presentan un rendimiento inferior, con tiempos de respuesta de 0.20, 0.28 y 0.41 segundos, respectivamente. Estos resultados evidencian que la ponderación del WRR mejora la eficiencia del sistema y mitiga los efectos del tráfico concurrente, especialmente en entornos de recursos desiguales.

- **Random:** En esta técnica, el controlador asigna las solicitudes entrantes a los servidores seleccionados aleatoriamente. Aunque su implementación es sencilla y no requiere mantener estados o estadísticas sobre los servidores, su eficiencia es limitada. La asignación al azar puede provocar desequilibrios, especialmente si algunos servidores reciben cargas mayores sin que tengan los recursos adecuados. En el contexto experimental de (Singh et al., 2022a; Vishwakarma, 2022), el

algoritmo Random presentó el peor rendimiento entre todas las técnicas evaluadas, con tiempos de respuesta más altos (0.41s) y menor eficiencia en la utilización de recursos.

- **Equal Load:** Esta técnica se basa en la distribución uniforme del tráfico, intentando asignar la misma cantidad de solicitudes a cada servidor sin importar su estado actual. Es útil en redes donde todos los nodos tienen características similares y el tráfico es constante. Aunque puede no ser tan eficiente como Least Connection o WRR en escenarios variables, Equal Load se mantiene estable y predecible. Es común en diseños básicos de balanceadores en SDN donde se prioriza la equidad y simplicidad sobre la adaptabilidad.

Tabla 1. *Comparación del Rendimiento de algoritmos Tradicionales de balanceo de carga en SDN*

Load Balancing Algorithm	Tiempo promedio de respuesta (sec)	Tasa de transacción (trans/sec)	Rendimiento (MB/sec)	Concurrencia
Weighted Round Robin	0.13	23.91	0.06	3.00
Round Robin	0.16	21.94	0.05	3.48
Equal Load	0.20	19.30	0.06	3.73
Least Connection	0.28	18.82	0.04	5.29
Random	0.41	16.46	0.04	6.71

Resumen de los resultados obtenidos al evaluar cinco algoritmos tradicionales de balanceo de carga en una red SDN.

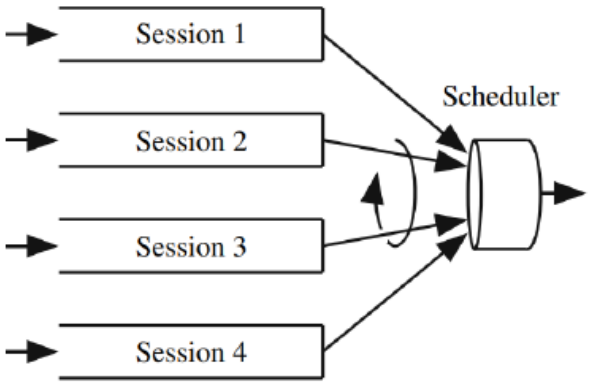
Fuente: (Vishwakarma, 2022)

Al integrarse en entornos SDN, las técnicas tradicionales de balanceo de carga aprovechan la visión global y el control centralizado del controlador, mejorando la asignación de recursos y los tiempos de respuesta. Sin embargo, su naturaleza estática y no adaptativa limita su eficacia en escenarios de tráfico dinámico, como redes en la nube o centros de datos distribuidos. Debido a su incapacidad para ajustarse en tiempo real ni tomar decisiones contextuales, han sido progresivamente reemplazadas por métodos inteligentes y dinámicos basados en aprendizaje automático y análisis predictivo

2.2.1.1 El protocolo Round Robin

El algoritmo Round Robin (RR) es una de las técnicas de balanceo de carga más antiguas, simples y utilizadas tanto en redes tradicionales como en arquitecturas definidas por software (SDN). Su funcionamiento se basa en una asignación cíclica y equitativa de las solicitudes entrantes a un conjunto de servidores disponibles. La facilidad de este algoritmo hace que RR sea ampliamente implementado en diversos entornos, especialmente cuando los servidores involucrados tienen características similares en términos de recursos y capacidad de procesamiento (Singh et al., 2022). La Figura 5 ilustra gráficamente el principio de funcionamiento secuencial del algoritmo, en el que múltiples sesiones se canalizan a través de un planificador que asigna de forma rotativa las solicitudes a los recursos disponibles.

Figura 5. *Funcionamiento de Round Robin*



Representación esquemática del principio de asignación cíclica de Round Robin.

Fuente: (Márquez Arpi, 2021)

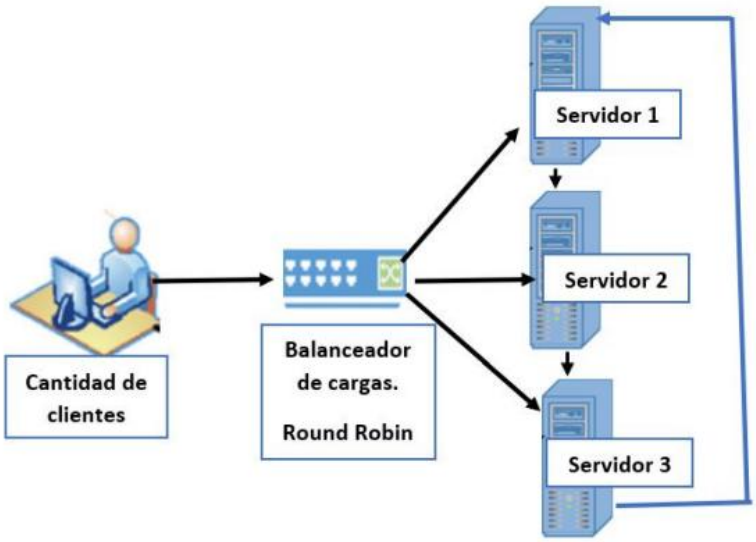
Round Robin (RR) es un algoritmo de asignación secuencial que distribuye solicitudes de forma cíclica entre servidores activos, sin considerar su estado de carga. Utiliza un índice que se incrementa con cada nueva petición, reasignando desde el inicio una vez que se llega al último servidor. Esta lógica, simple y predecible, lo convierte en una solución de bajo costo computacional (Putra & Purnomo, 2022)

Su desempeño es óptimo en entornos homogéneos, donde los servidores tienen capacidades similares y se manejan tipos de tráfico uniformes, como HTTP o DNS. En estos casos, asegura una distribución equilibrada y estable (Vishwakarma, 2022). Sin embargo, al no evaluar la carga en tiempo real, puede asignar nuevas solicitudes a servidores ya saturados, provocando subutilización de recursos disponibles y degradación del rendimiento en contextos con alta variabilidad (Permana Arrasyid et al., 2024).

En redes SDN, Round Robin opera en el plano de control, donde el controlador administra el orden rotativo de asignación. Esta lógica puede integrarse mediante políticas de reenvío o scripts, permitiendo aplicar su comportamiento dentro de reglas dinámicas

de flujo. La Figura 6 ilustra este proceso, donde múltiples clientes son atendidos por servidores mediante un balanceador que sigue la lógica cíclica del algoritmo.

Figura 6. *Aplicación del algoritmo Round Robin en el balanceo de carga*



Fuente: (Vásquez Figueroa, 2022)

El algoritmo Round Robin (RR) se caracteriza por su sencillez y naturaleza cíclica, lo que facilita su implementación en entornos homogéneos. Sin embargo, su enfoque estático representa una limitación en redes con alta variabilidad, ya que no considera parámetros como la carga actual, el uso de CPU o la latencia de los servidores, generando asignaciones ineficientes y posibles desequilibrios de carga (Zhou & Liu, 2024). En entornos heterogéneos, esta falta de adaptabilidad puede provocar cuellos de botella al asignar tareas exigentes a nodos de menor capacidad. Dentro de arquitecturas SDN, donde la gestión dinámica y la calidad de servicio son esenciales, el RR resulta útil solo en escenarios controlados, pero su ausencia de respuesta en tiempo real limita su desempeño ante demandas fluctuantes y tráfico no uniforme.

A pesar de estas limitaciones, investigaciones como la de (Singh et al., 2022) han demostrado que, en pruebas controladas con el controlador POX y hasta 255 usuarios concurrentes, el algoritmo mantuvo una disponibilidad del 100% y tiempos de respuesta inferiores a un segundo. La Tabla 2 resume estos resultados, evidenciando que Round Robin puede ofrecer un rendimiento aceptable bajo ciertas condiciones de carga estable y controlada.

Tabla 2. *Rendimiento del algoritmo en balanceo de carga*

Usuarios Concurrentes	**Transacciones**	**Tiempo de respuesta promedio (s)**	**Disponibilidad (%)**
10 a 255	10,752 a 18,950	0.06 a 0.80	100

Los resultados evidencian que Round Robin (RR) puede manejar grandes volúmenes de tráfico en infraestructuras homogéneas sin afectar la calidad del servicio. Sin embargo, su simplicidad limita su efectividad en escenarios complejos. No considera el estado dinámico de los servidores (carga, CPU o latencia), lo que puede generar asignaciones ineficientes y desequilibrio de carga (Vishwakarma, 2022).

Como respuesta, se han desarrollado variantes como Weighted Round Robin (WRR) y W-RRLC, que integran ponderaciones o monitoreo de conexiones activas para mejorar el reparto de carga en entornos heterogéneos (Permana Arrasyid et al., 2024). A pesar de ello, RR continúa utilizándose en sistemas distribuidos y servidores web por su fácil implementación, aunque su efectividad se reduce ante necesidades actuales más complejas.

2.2.2 Técnicas de balanceo de carga con redes neuronales en redes SDN

El uso de redes neuronales en el ámbito del balanceo de carga para redes definidas por software (SDN) ha ganado creciente interés debido a su capacidad para adaptarse dinámicamente a los cambios en el comportamiento del tráfico. Estas redes inteligentes permiten superar las limitaciones de los algoritmos tradicionales, ofreciendo predicciones más precisas, control eficiente del tráfico, y toma de decisiones en tiempo real basadas en métricas contextuales.

En SDN, el controlador actúa como el centro de decisiones, y la incorporación de modelos de redes neuronales dentro de este plano de control permite automatizar la distribución de flujos y anticiparse a la aparición de congestiones. A continuación, se describen en detalle las principales arquitecturas de redes neuronales empleadas en los estudios recientes para tareas de balanceo de carga en SDN:

- **Redes Neuronales Artificiales (ANN):** Son modelos de aprendizaje supervisado que pueden identificar patrones complejos a partir de múltiples métricas de red. Según (Gowri & Baranidharan, 2025), las ANN procesan entradas como latencia, pérdida de paquetes y ancho de banda para determinar la mejor ruta de reenvío. Estas decisiones son enviadas al controlador SDN para la actualización de las reglas de flujo. Su implementación ha demostrado mejoras significativas en reducción de retardo y jitter.
- **Redes Neuronales Convolucionales (CNN):** Utilizadas principalmente para predecir zonas de congestión en centros de datos SDN. Según (Alhilali & Montazerolghaem, 2023a), una CNN puede aprender del comportamiento de tráfico y topologías anteriores para anticipar condiciones críticas y redistribuir

tráfico de forma proactiva. Esto mantiene la calidad de servicio incluso ante picos de demanda.

- **Redes Neuronales Recurrentes (RNN):** Adecuadas para analizar secuencias temporales, las RNN pueden predecir el estado futuro de los nodos en base al historial de tráfico. (Gowri & Baranidharan, 2025)implementan una RNN para identificar el nodo menos congestionado a futuro, lo que permite redirigir los flujos antes de que se presenten cuellos de botella.
- **Perceptrones Multicapa (MLP):** Estas redes neuronales profundas permiten capturar relaciones no lineales entre variables como carga de CPU, uso de memoria o densidad de tráfico. (Zakia & Ben Yedder, 2017) evaluaron su desempeño en escenarios de tráfico irregular, demostrando que los MLP mantienen el retardo por debajo de los 30 ms y distribuyen eficientemente la carga.
- **Redes LSTM (Long Short-Term Memory):** Una extensión de las RNN, las LSTM son capaces de mantener información relevante durante secuencias largas, evitando la pérdida de contexto. (Alhilali & Montazerolghaem, 2023a)demostraron que las LSTM pueden reducir el tiempo de reconfiguración de rutas en más de un 35% respecto a métodos tradicionales, gracias a su precisión en la predicción de tráfico futuro.

A modo de resumen, se presenta la siguiente tabla comparativa que permite visualizar las características principales, ventajas y limitaciones de cada modelo de red neuronal aplicado al balanceo de carga en SDN:

Tabla 3. *Comparación de modelos de redes neuronales*

Modelo de Red Neuronal	Aplicación en SDN	Ventajas	Limitaciones Principales
ANN	Selección de rutas óptimas	Fácil de implementar, buena para tareas supervisadas	Menor precisión ante patrones temporales complejos
CNN	Predicción de congestiones	Buena para detección espacial y reconocimiento de patrones	Alta demanda computacional
RNN	Predicción de tráfico a futuro	Manejo de datos secuenciales, memoria temporal	Problemas con secuencias largas (desvanecimiento)
MLP	Distribución de tráfico irregular	Captura relaciones no lineales complejas	Necesita grandes volúmenes de datos para entrenar
LSTM	Reconfiguración anticipada de rutas	Memoria de largo plazo, precisión en predicción dinámica	Más costosa computacionalmente

2.2.2.1 Modelo Long Short-Term Memory

En el contexto de las redes definidas por software (SDN), el modelo Long Short-Term Memory (LSTM) se ha consolidado como una herramienta clave para la gestión predictiva del tráfico y el balanceo de carga, debido a su capacidad de modelar series temporales con dependencias de largo plazo. A diferencia de los métodos tradicionales, el LSTM puede capturar patrones históricos del tráfico y utilizarlos para anticipar condiciones futuras de congestión o variación en la demanda, lo que mejora la toma de decisiones del controlador SDN en tiempo real. Este modelo, propuesto por Hochreiter y Schmidhuber (1997), extiende la arquitectura de las redes neuronales recurrentes (RNN) al incorporar una celda de memoria interna y mecanismos de compuertas (entrada, olvido y salida) que controlan el flujo de información. Dicho diseño mitiga los problemas de desvanecimiento y explosión del gradiente, permitiendo un aprendizaje estable en

secuencias largas y una administración más precisa de los recursos de red mediante predicción adaptativa.

El funcionamiento de un bloque LSTM se basa en la interacción de tres puertas fundamentales:

- **Puerta de olvido (Forget Gate):** controla qué parte del estado anterior de la celda debe ser descartada. Se calcula como: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ en donde :

  a) x_t : Entrada en el tiempo *t,* por ejemplo, el uso de CPU, latencia o tráfico

  b) h_{t-1}: Salida del paso anterior, el cual proporciona contexto reciente

  c) W_f: Pesos asociados a la puerta de olvido

  d) b_f: Sesgo correspondiente

  e) σ: Función sigmoide, que devuelve valores entre 0 y 1

  f) f_t: Fracción de la memoria anterior C_{t-1}

- **Puerta de entrada (Input Gate):** determina qué nueva información se incorporará al estado de la celda: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ y $\widetilde{C_t} = \tan h(W_C \cdot [h_{t-1}, x_t] + b_C)$ en donde:

  a) W_i, b_i: Pesos y sesgos de la puerta de entrada

  b) i_t: Indica cuánta información nueva se integrará

  c) $\widetilde{C_t}$: Candidato a nueva información útil para la memoria

  d) W_C, b_C: Pesos y sesgos para calcular $\widetilde{C_t}$

  e) Tanh: Función de activación que permite representar valores entre -1 y 1

- **Actualización del estado de la celda:** la información retenida y la nueva información se combinan para actualizar el estado de la celda: $C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$ en donde:
    a) C_{t-1}: Memoria previa de largo plazo
    b) C_t= Nueva memoria de celda, resultado de conservar lo útil del pasado y sumar lo nuevo

- **Puerta de salida (Output Gate):** regula qué parte de la información será emitida como salida del bloque LSTM: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) h_t = o_t * \tan h\,(C_t)$
- W_o, b_o: Pesos y sesgos de la puerta de salida
- o_t: Indica que parte se convierte en salida
- h_t: Salida final del bloque en el instante *t*, también se convierte en entrada para el siguiente paso

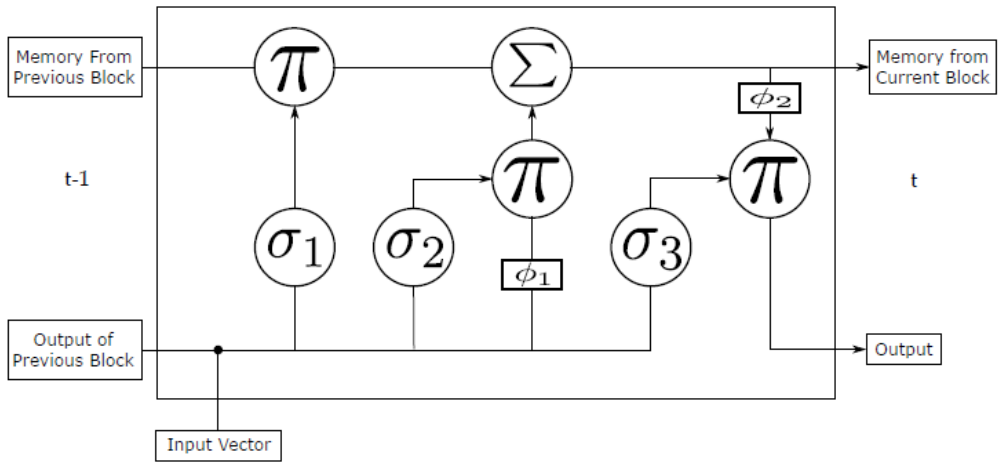
La **Figura 7** ilustra el flujo de datos y operaciones internas de un bloque LSTM típico, destacando cómo se combinan las entradas, salidas y memorias entre instantes de tiempo consecutivos para procesar secuencias. Esta arquitectura modular permite al LSTM mantener información a largo plazo de forma controlada y adaptativa.

En el bloque se observan:

- **Entrada de datos actual (Input Vector)**: es el dato nuevo x_t que se incorpora al modelo en el instante actual.
- **Salida del paso anterior (Output of Previous Block)**: corresponde a h_{t-1}, salida previa que ayuda a contextualizar la entrada.

- **Memoria del paso anterior (Memory From Previous Block)**: es , la celda de memoria que acumula conocimientos pasados.
- **Puertas sigmoides (σ)**: representan las funciones sigmoides de las puertas de olvido, entrada y salida, respectivamente. Estas puertas deciden qué parte de la información debe conservarse, actualizarse o salir del bloque.
- **Multiplicaciones**: indican el control sobre el flujo de datos mediante operaciones de compuerta
- **Candidatos de memoria:** modelan la transformación de la nueva información mediante funciones tangente hiperbólica.
- **Suma ($\sum$)**: representa la actualización del estado de la celda, combinando lo que se retiene del pasado y lo nuevo que se agrega.
- **Salidas (Memory from Current Block y Output)**: son respectivamente C_t y h_t, que serán utilizados en el siguiente paso de tiempo.

Figura 7. *Representación del funcionamiento de interno de LSTM*



Fuente: (Kumar et al., 2018)

Los modelos LSTM son entrenados utilizando datasets recolectados por el controlador, que incluyen métricas como uso de CPU, carga de enlace, pérdida de paquetes y tasa de llegada de tráfico. Estos datos se estructuran como series temporales y se utilizan para predecir condiciones futuras de red. Una vez entrenado, el LSTM puede recibir datos en tiempo real y devolver predicciones sobre el estado de cada nodo o enlace en los próximos intervalos.

En el estudio de (Simaiya et al., 2024), se propuso un sistema híbrido basado en CNN-LSTM optimizado con algoritmos evolutivos como PSO (Particle Swarm Optimization) y GA (Genetic Algorithm). En esta arquitectura, la CNN se encargaba de extraer características espaciales de la topología de red, mientras que el LSTM capturaba la evolución temporal de las métricas de tráfico. Esta combinación permitió una predicción precisa y rápida de puntos de congestión, logrando reducir el tiempo de respuesta en un 28% y mejorar el rendimiento general del sistema SDN.

Ventajas del uso de LSTM frente a otros enfoques

- **Predicción anticipada:** el LSTM puede detectar con anticipación patrones de congestión y redirigir el tráfico hacia rutas alternativas, mejorando la calidad del servicio (QoS).
- **Aprendizaje de comportamiento complejo:** su capacidad para retener memoria de largo plazo le permite comprender patrones que ocurren en intervalos no regulares, algo crucial en redes con tráfico volátil.
- **Robustez en entornos dinámicos:** gracias a su adaptabilidad, los modelos LSTM se comportan de forma más estable en redes con alta variabilidad de tráfico, como las redes en la nube, redes móviles o infraestructuras IoT.

Métricas de evaluación en modelos LSTM aplicados a redes

Los modelos LSTM aplicados al balanceo de carga en redes SDN se evalúan mediante múltiples métricas, entre ellas:

- **Tiempo de respuesta promedio**
- **Throughput o tasa de transferencia efectiva**
- **Utilización de recursos (CPU, RAM, ancho de banda)**
- **Precisión de predicción (RMSE, MAE)**
- **Ratio de pérdida de paquetes (%)**
- **Eficiencia energética total (en kWh)**

Estas métricas permiten medir tanto el rendimiento del modelo como el impacto directo sobre la experiencia del usuario final.

2.3 Herramientas de Software SDN

El desarrollo y la validación de arquitecturas de Redes Definidas por Software (SDN) requieren entornos de experimentación que permitan evaluar el comportamiento del plano de control y del plano de datos sin los altos costos ni los riesgos de manipular hardware físico en producción. Para mitigar esta barrera, la comunidad investigativa se apoya en diversas plataformas de software especializadas, abarcando desde emuladores de alto rendimiento orientados a imágenes de red comerciales (como GNS3 y EVE-NG) y simuladores de eventos discretos para el modelado matemático riguroso (como NS-3 y OMNeT++), hasta herramientas de emulación basada en virtualización ligera como Mininet; esta última consolidada como el estándar de facto en la investigación SDN, dado

que permite la ejecución de topologías complejas con un *stack* de red real y facilita la interacción transparente con controladores externos mediante el protocolo OpenFlow.

2.3.1 Plataformas de simulación de redes

- **Mininet**: es un emulador ligero de código abierto que permite crear redes virtuales completas (hosts, switches, controladores) en un solo equipo físico. Utiliza virtualización de sistema operativo y es ampliamente usado para simular redes SDN con controladores como Ryu o Floodlight. Su compatibilidad con OpenFlow y su capacidad para medir rendimiento lo convierten en una herramienta efectiva para probar algoritmos de balanceo como Round Robin o RNN en entornos virtualizados (Alonso et al., 2020; Amaya Fariño et al., 2022; Li et al., 2020).

- **GNS3**: por su parte, se centra en la emulación precisa de dispositivos reales como routers y switches mediante imágenes reales de sistemas operativos (Cisco IOS, JUNOS, etc.). Su arquitectura cliente-servidor y compatibilidad con dispositivos físicos lo hacen ideal para redes híbridas. Aunque su consumo de recursos puede ser alto, es muy útil para probar protocolos de enrutamiento y configuraciones reales en un entorno controlado (Golightly et al., 2023; Smera & Sandeep, 2022).

- **EVE-NG**: destaca por su capacidad de emular topologías complejas usando imágenes reales de firmware. Soporta múltiples dispositivos de distintos fabricantes (Cisco, Juniper, Palo Alto), integración con herramientas como Wireshark y colaboración multiusuario. Es altamente valorado en educación y formación avanzada por su interfaz web intuitiva y capacidad de simular redes cercanas a la realidad (Climent Fornés, 2023; Harahus et al., 2023; Sharma et al., 2024).

- **NS-3:** es una herramienta de simulación basada en eventos discretos, muy utilizada en investigación por su precisión y escalabilidad. Su arquitectura

modular permite definir nodos, canales, protocolos y analizar el tráfico con herramientas como Wireshark. Además, se integra con inteligencia artificial a través de extensiones como ns-3-gym, ideal para probar algoritmos de aprendizaje en redes complejas (Campanile et al., 2020; Khanderao & Khedekar, 2021; Kuse & Jaeger, 2020)

- **OMNeT++:** es un simulador modular con enfoque general, utilizado para redes, protocolos, y sistemas distribuidos. Su entorno gráfico, lenguaje NED y frameworks como INET o MiXiM permiten modelar redes IP, inalámbricas, vehiculares y de sensores. A pesar de su complejidad, su capacidad de visualización en tiempo real lo hace ideal para experimentar con algoritmos de balanceo de carga en entornos visuales y jerárquicos (Bachmeier et al., 2020; Barbecho Bautista et al., 2020; Shaik & Kim, 2025)

Cada simulador posee características particulares en cuanto a arquitectura, compatibilidad, recursos requeridos y nivel de realismo. Por ello, la Tabla 4 presenta una comparación detallada de cinco plataformas ampliamente utilizadas en investigación: Mininet, GNS3, EVE-N G, NS-3 y OMNeT++. Esta comparación considera criterios técnicos esenciales como el tipo de simulación, facilidad de configuración, consumo de recursos, soporte para inteligencia artificial y adecuación al estudio de algoritmos como Round Robin y redes neuronales recurrentes (RNN). El objetivo es identificar la herramienta más idónea para implementar pruebas en entornos virtualizados orientados a la optimización del tráfico mediante técnicas tradicionales e inteligentes.

Tabla 4. *Comparación de Simuladores de red*

Simulador	Tipo de Simulación	Orientación Principal	Lenguaje Principal	Visualización Gráfica	Consumo de Recursos	Curva de Aprendizaje	Escenarios para Balanceo de Carga	Más eficiente para balanceo de carga
Mininet	Emulación ligera en tiempo real (basado en namespaces)	Redes SDN y OpenFlow	Python (scripts CLI)	Limitada	Bajo	Baja	Ideal para SDN con Ryu y Floodlight	Sí, si se enfoca en controladores SDN
GNS3	Emulación con imágenes reales (Cisco, etc.)	Protocolos de capa 2/3, educación y certificaciones	Configuración gráfica + CLI	Alta (GUI completa)	Alto	Media	Compatible, pero limitado para IA	No ideal para RNN, mejor para RR clásico
EVE-NG	Emulación con VMs y múltiples proveedores	Simulación profesional con múltiples VMs	Interfaz web (HTML5) + CLI	Muy alta (interfaz web)	Alto	Media-Alta	Muy útil si se integran controladores en VM	Sí, pero requiere recursos potentes
NS-3	Simulación basada en eventos discretos	Investigación académica de protocolos de red	C++ y Python	Limitada	Medio	Alta	Potente con ns3-gym para IA	Sí, especialmente si se integra con IA
OMNeT++	Simulación basada en eventos discretos	Simulación modular de redes y sistemas distribuidos	C++ y NED	Avanzada (TkEnv/Qtenv)	Medio-Alto	Alta	Potente con INET, ideal para visualizar tráfico	Sí, si se necesita visualización detallada y tráfico complejo

2.3.2 Controladores SDN

En el entorno de las redes definidas por software (SDN), los controladores son componentes importantes que gestionan la lógica de red desde el plano de control. A continuación, se resumen las principales características de los controladores más utilizados en investigación y simulación de algoritmos como Round Robin y RNN.

- **RYU:** es un controlador de código abierto desarrollado en Python por NTT, ideal para entornos educativos y de experimentación. Su arquitectura modular y su compatibilidad con versiones recientes de OpenFlow lo hacen muy flexible. Se integra fácilmente con Mininet y permite realizar pruebas de rendimiento como latencia, jitter y throughput, aunque puede verse afectado por el aumento de nodos y eventos packet_in (Bhardwaj & Panda, 2022; Islam et al., 2020)

- **OpenDaylight (ODL):** es una plataforma SDN robusta basada en Java, promovida por la Linux Foundation. Ofrece gran modularidad a través de su capa MD-SAL, soporte para múltiples protocolos, y ha evolucionado hacia funciones avanzadas como NFV y seguridad TLS. A pesar de su potencia, su configuración puede ser compleja y requiere mayor capacidad de hardware, especialmente desde la eliminación de módulos clave en versiones recientes (Aditi et al., 2022; Badotra & Panda, 2020)

- **POX**: derivado de NOX, es un controlador ligero y monohilo desarrollado en Python. Su simplicidad y bajo consumo de recursos lo hacen ideal para prototipos y entornos académicos. Aunque limitado a OpenFlow 1.0 y sin soporte para multithreading, destaca por su baja latencia y facilidad de uso en simulaciones pequeñas con tráfico controlado (Elmoslemany et al., 2020; Mohammad Nowsin Amin Sheikh et al., 2019; Noman & Jasim, 2020).

- **ONOS:** está orientado a redes de gran escala, con una arquitectura distribuida y tolerante a fallos. Soporta clustering entre múltiples nodos, lo que mejora su rendimiento en términos de latencia, jitter y throughput. Además, ofrece una API REST, interfaz visual y compatibilidad con protocolos modernos, siendo ideal para redes productivas y simulaciones complejas (Septian et al., 2021; Zhu et al., 2019b).

Cada herramienta de simulación presenta características técnicas distintas en cuanto a tipo de emulación, facilidad de configuración, nivel de realismo, consumo de recursos y soporte para tecnologías avanzadas como inteligencia artificial. En este contexto, la Tabla 5 presenta una comparación detallada de cinco plataformas ampliamente utilizadas en investigación: Mininet, GNS3, EVE-NG, NS-3 y OMNeT++. Esta tabla permite identificar las fortalezas y limitaciones de cada simulador, facilitando la selección del entorno más adecuado para realizar pruebas experimentales dentro de redes SDN virtualizadas.

Tabla 5. *Comparación de Controladores*

Controlador	Lenguaje de programación	Arquitectura	Soporte OpenFlow	Consumo de recursos	Rendimiento general	Limitaciones principales	Soporte para Inteligencia Artificial
POX	Python	Centralizada, monohilo	1.0	Muy bajo	Bajo en topologías grandes	No soporta clustering, no escalable	Bajo: no modular, sin soporte nativo
RYU	Python	Modular, centralizada	1.0 – 1.5	Bajo	Medio	No distribuido, limitado en redes grandes	Medio: permite integración directa con Python
OpenDaylight	Java	Modular, extensible	1.0 – 1.5+	Alto	Alto (según módulos activos)	Complejidad alta, consumo elevado	Medio: requiere APIs externas o módulos
ONOS	Java	Distribuida (clúster)	1.0 – 1.5+	Medio – Alto	Muy alto en escenarios amplios	Requiere configuración avanzada y cluster	Alto: soporta aplicaciones IA en clúster

2.4 **Modelo de Rendimiento UF1880**

En el presente proyecto de tesis, se requiere de un modelo de referencia que permita evaluar con objetividad el comportamiento de ambos enfoques bajo escenarios controlados.

El Modelo de Rendimiento UF1880 se adapta a esta necesidad porque proporciona una metodología estructurada por fases, desde la definición de herramientas hasta la obtención de métricas cuantitativas, que permiten determinar cuál de los dos algoritmos ofrece un mejor desempeño en términos de eficiencia en la distribución de tráfico, reducción de latencia y aprovechamiento de recursos dentro de una red SDN.

De esta manera, el modelo no se limita a describir indicadores genéricos, sino que se articula de forma práctica con el proceso experimental que guía la comparación entre Round Robin, caracterizado por su simplicidad, y las RNN, cuyo aprendizaje secuencial permite anticipar patrones de tráfico y optimizar la toma de decisiones en el balanceo de carga (Menédez Arantes Silvia Clara, 2019).

En la **fase 0** se establece la selección de las herramientas de software y la construcción de los escenarios experimentales. Para ello se recurre al uso de GNS3, que permite emular topologías SDN de diversa complejidad, y al controlador RYU, en el cual se implementa la lógica tanto de Round Robin como de la RNN. En paralelo, se emplean librerías de Python y TensorFlow para el desarrollo y entrenamiento del modelo neuronal, con el propósito de que pueda aprender patrones de tráfico y anticipar situaciones de congestión. Esta fase garantiza que ambos algoritmos se ejecuten en un mismo entorno estandarizado, eliminando sesgos y asegurando que la comparación se realice bajo condiciones idénticas.

La **fase 1** comprende la implementación del entorno experimental y la ejecución inicial de los algoritmos. En este punto se configuran las topologías definidas en GNS3 y se integran con el controlador RYU. Primero se implementa el protocolo Round Robin como referencia estática, distribuyendo el tráfico de manera equitativa entre los enlaces. Posteriormente, se activa el módulo de RNN, que, gracias a su capacidad de aprendizaje secuencial, ajusta dinámicamente las decisiones de balanceo. Durante esta fase se generan flujos de tráfico TCP/IP controlados que permiten observar cómo cada enfoque enruta los paquetes, tanto en condiciones de carga normal como en escenarios de saturación.

En la **fase 2** se definen y analizan las métricas de evaluación que permitirán establecer una comparación objetiva entre Round Robin y RNN. Los indicadores seleccionados incluyen throughput, ancho de banda utilizado, jitter, tiempo de respuesta y pérdida de paquetes. Estas métricas proporcionan una visión integral del desempeño de la red, ya que abarcan desde la eficiencia en la transmisión de datos hasta la calidad de la experiencia percibida por el usuario. Gracias a ellas, es posible determinar con precisión si la simplicidad del Round Robin resulta suficiente en un entorno SDN, o si la adaptabilidad de las RNN ofrece un beneficio tangible en términos de reducción de latencia, aprovechamiento de recursos y estabilidad del tráfico.

Finalmente, en la **fase 3** se lleva a cabo la evaluación de los parámetros definidos mediante el uso de herramientas de análisis y validación. Entre estas se destaca iPerf, que permite medir throughput, ancho de banda y pérdidas de paquetes de forma precisa; Ping, empleado para calcular tiempos de respuesta y tasas de pérdida de conectividad; y Wireshark, utilizado como sniffer para capturar tráfico y analizar estadísticas detalladas como jitter y variación en los retardos. Esta fase culmina con la obtención de resultados estadísticos que sustentan la comparación entre Round Robin y RNN, ofreciendo una base empírica sólida para las conclusiones del proyecto de tesis.

CAPÍTULO III: DISEÑO E IMPLEMENTACIÓN

Se pretende realizar un estudio comparativo entre dos enfoques de balanceo de carga en redes definidas por software (SDN), por un lado, a través del protocolo estático Round Robin y por otro las redes neuronales recurrentes (RNN). El desarrollo del proyecto sigue la metodología en cascada, permitiendo una secuencia clara y estructurada de las fases del proyecto. Este enfoque metodológico asegura que cada etapa se complete de manera secuencial y lógica, permitiendo que los resultados obtenidos en una fase sirvan como base para el desarrollo de la siguiente.

3.1.1 Justificación del uso de la metodología en cascada

La elección de la metodología en cascada se justifica por la interdependencia técnica lineal de los componentes del sistema. En el desarrollo de soluciones de balanceo de carga con Inteligencia Artificial, existe una dependencia estricta donde es indispensable completar la recolección de *datasets* y el entrenamiento de la red neuronal (Fase de Diseño) antes de su despliegue operativo en el controlador SDN (Fase de Implementación).

Esta estructura secuencial es crítica para la validez experimental, ya que permite aislar la fase de pruebas finales. De este modo, se asegura que la comparación de métricas de rendimiento (Round Robin vs. LSTM) se ejecute sobre un sistema estable y con parámetros definidos, evitando modificaciones en el código durante la recolección de datos que pudieran alterar los resultados de los escenarios de congestión.

3.1.2 Fases del ciclo de vida aplicadas al proyecto

El desarrollo del sistema se estructura en cuatro fases secuenciales alineadas con la metodología en cascada. Para garantizar la rigurosidad en la evaluación de la red, se integran transversalmente las directrices del Modelo de Rendimiento UF1880 (Gestión de Servicios y Control de Rendimiento), el cual establece los protocolos para la recolección de métricas, el establecimiento de líneas base y el análisis de desviaciones en la infraestructura tecnológica.

Fase 1: Análisis de Requisitos y Definición de Línea Base. En esta etapa se establecen los parámetros operativos del entorno SDN. Siguiendo el modelo UF1880, se define la línea base de rendimiento (*baseline*) del sistema, identificando los indicadores críticos a monitorizar. Se seleccionan las métricas de Calidad de Servicio (QoS) estandarizadas en la literatura técnica: *throughput*, latencia de extremo a extremo y tasa de pérdida de paquetes. Esta fase asegura que los criterios de comparación sean objetivos y medibles.

Fase 2: Diseño del Sistema. Se modela la arquitectura de la solución en dos frentes. Primero, se diseña una topología de red tipo *Fat-Tree* para garantizar la redundancia de rutas necesaria para el balanceo. Segundo, se define la arquitectura de la Red Neuronal Recurrente (LSTM), determinando sus capas de entrada (uso de CPU, ancho de banda, tiempo de respuesta) y salida, diseñadas para predecir la saturación de los recursos identificados en la fase de análisis.

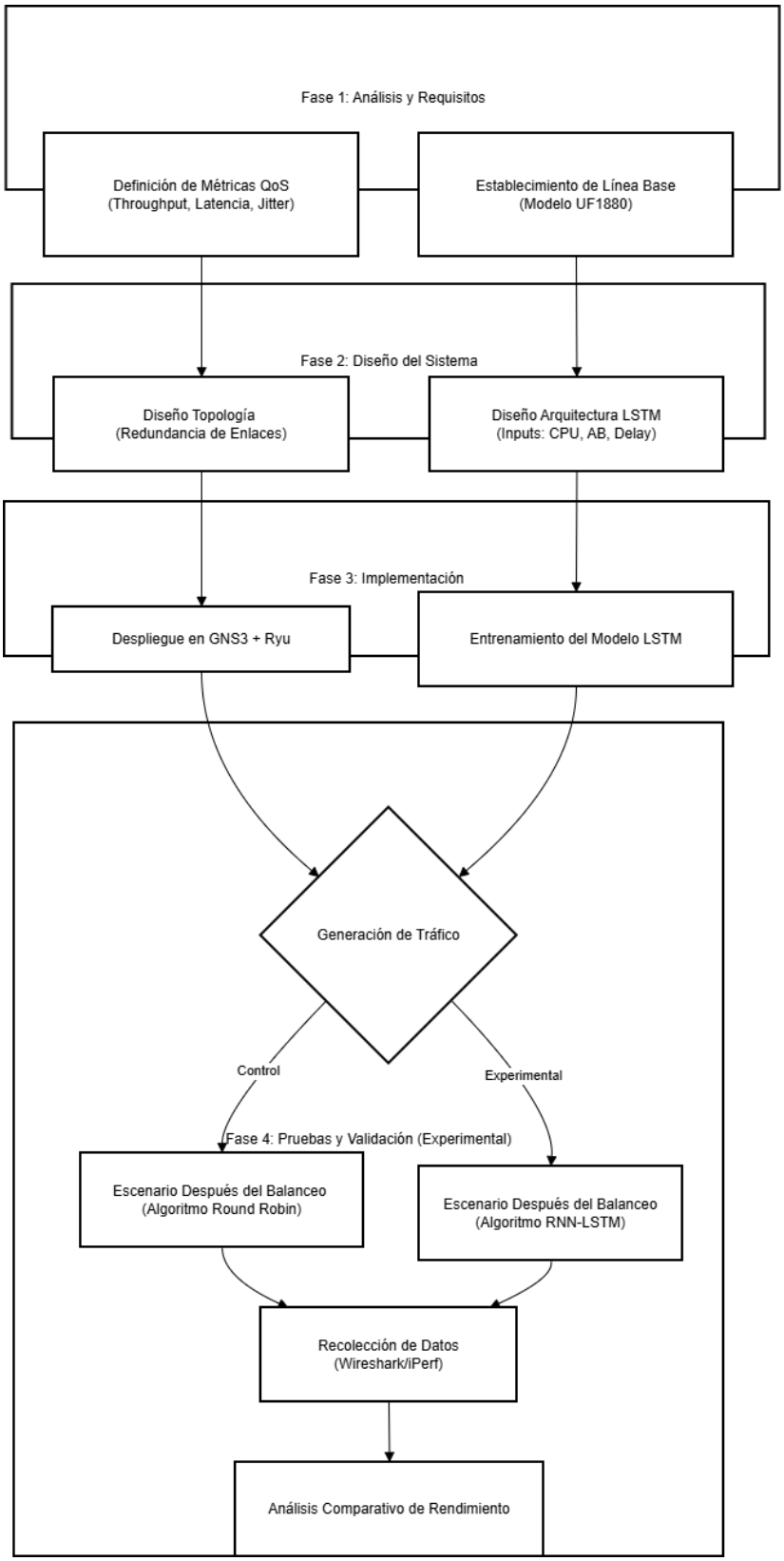
Fase 3: Implementación. Abarca la codificación de los algoritmos en Python y el despliegue en el entorno de emulación Mininet. Se realiza el entrenamiento del modelo LSTM y la programación del controlador Ryu. En esta fase se implementan los agentes de monitorización requeridos por el modelo de rendimiento para la recolección continua

de datos mediante el protocolo OpenFlow 1.3, traduciendo las predicciones de la IA en reglas de flujo (*Flow Entries*) instaladas en los *switches* virtuales.

Fase 4: Pruebas y Validación (Monitorización y Ajuste). Finalmente, se ejecutan las simulaciones aplicando el ciclo de control de rendimiento. Se definen dos escenarios: *Before Load Balancing* (BLB) con Round Robin y *After Load Balancing* (ALB) con LSTM. Se recolectan estadísticas en tiempo real para contrastarlas contra la línea base definida en la Fase 1, permitiendo cuantificar las desviaciones positivas (mejoras) en el rendimiento y validar la hipótesis de investigación.

La interacción entre estas etapas y los puntos de control del modelo de rendimiento se ilustra detalladamente en la Figura 8, que presenta el diagrama de flujo metodológico del proyecto.

Figura 8. *Fases del Proyecto*



3.2 **Análisis y Requisitos del Sistema**

En este apartado se describe el sistema propuesto, los requisitos que debe cumplir y las condiciones necesarias para su implementación. El objetivo es demostrar que la solución es viable y definir de forma clara qué debe hacer el sistema y bajo qué restricciones operará.

3.2.1 Estudio de viabilidad

El proyecto se fundamenta en herramientas de software ampliamente validadas en la comunidad científica para la investigación en redes y el desarrollo de aplicaciones SDN. Para la emulación de la infraestructura de red se utilizará Mininet, una plataforma que permite crear topologías de red realistas utilizando virtualización ligera basada en procesos (namespaces) sobre un único kernel de Linux. A diferencia de los simuladores tradicionales, Mininet instancia conmutadores virtuales (como Open vSwitch) y hosts que ejecutan código real, ofreciendo soporte nativo para el protocolo OpenFlow y posibilitando la creación de un escenario controlado para el estudio del balanceo de carga.

Como plano de control se emplea el controlador Ryu, un framework de código abierto diseñado específicamente para la gestión de redes SDN. Ryu proporciona una robusta API en Python que facilita la implementación de la lógica de balanceo y la comunicación fluida con la topología desplegada en Mininet. Esta integración permite al controlador interactuar dinámicamente con los switches para instalar reglas de flujo (Flow Mods) y monitorear estadísticas de tráfico en tiempo real. Gracias a esta arquitectura, es factible integrar y comparar tanto el algoritmo Round Robin como el modelo predictivo basado en redes neuronales recurrentes (RNN/LSTM) de manera transparente, sin necesidad de alterar el hardware físico.

En cuanto a la viabilidad técnica, el equipamiento disponible en los laboratorios de Ingeniería en Telecomunicaciones resulta idóneo para ejecutar este entorno. Dado que Mininet optimiza el uso de CPU y memoria mediante virtualización a nivel de sistema operativo, permite replicar escenarios de prueba complejos y repetibles con un consumo de recursos significativamente menor que otras soluciones de virtualización completa. Esto garantiza la capacidad de ejecutar múltiples iteraciones experimentales bajo distintas condiciones de estrés de red, asegurando la validez estadística del análisis comparativo propuesto sin poner en riesgo infraestructuras de producción.

carga en redes SDN.

3.2.2 Requisitos funcionales y no funcionales

En esta sección se definen los requisitos funcionales del sistema propuesto, es decir, las funciones específicas que debe realizar el controlador SDN desarrollado sobre Ryu para implementar y evaluar los algoritmos de balanceo de carga Round Robin y LSTM en entornos simulados con Mininet. La Tabla 6 se resume cada requisito, proporcionando un identificador y una descripción clara orientada a los objetivos experimentales de la tesis.

Tabla 6. *Requisitos funcionales*

ID	Descripción
RF1	El sistema debe implementar un controlador SDN basado en Ryu capaz de recibir y procesar eventos de la red, como mensajes de descubrimiento y PacketIn, para gestionar dinámicamente el encaminamiento del tráfico.
RF2	El sistema debe implementar un módulo de balanceo de carga con algoritmo Round Robin que distribuya los flujos entre un conjunto de servidores backend definidos en la topología SDN.
RF3	El sistema debe implementar un módulo de balanceo de carga basado en redes neuronales recurrentes (LSTM), que seleccione servidor o ruta en función del estado de la red y de las métricas de tráfico recolectadas por el controlador.

RF4	El sistema debe permitir cambiar entre el modo Round Robin y el modo LSTM sin modificar la topología física o lógica de la red, con el fin de facilitar la comparación experimental entre ambos enfoques.
RF5	El sistema debe recolectar y almacenar métricas de desempeño, como utilización de enlaces, retardo, pérdida de paquetes y throughput por servidor, para cada escenario de prueba definido.

Además de las funciones que debe realizar el sistema, es necesario especificar los requisitos no funcionales, los cuales describen las características de calidad que debe cumplir la solución propuesta estos requisitos se relacionan con el rendimiento, escalabilidad, confiabilidad, mantenibilidad y uso de herramientas de software libre en la implementación del controlador SDN. La Tabla 7 proporciona los requisitos no funcionales considerados.

Tabla 7. *Requisitos no funcionales*

ID	Descripción
RNF1	El controlador SDN debe gestionar el volumen de flujos definido en los escenarios experimentales sin provocar una degradación significativa en el tiempo de instalación de reglas ni en el tiempo de respuesta de la red.
RNF2	El sistema debe mantener precisión en la medición de métricas de red (tráfico, retardo, pérdida de paquetes y throughput), de manera que los resultados de la comparación entre algoritmos sean reproducibles y confiables.
RNF3	El sistema debe ser escalable a distintas topologías y cantidades de servidores dentro del entorno de simulación GNS3, únicamente en la definición de la topología.

RNF4	El tiempo requerido para cambiar entre los algoritmos de balanceo (Round Robin y LSTM) y para configurar nuevos escenarios experimentales debe ser razonablemente bajo, facilitando la ejecución de múltiples pruebas.
RNF5	El sistema debe emplear exclusivamente herramientas y librerías de software libre u open source (por ejemplo, Mininet, Ryu y frameworks de deep learning), evitando la dependencia de licencias comerciales.
RNF6	La arquitectura del sistema debe ser modular, separando la lógica de descubrimiento de topología, balanceo de carga y recolección de métricas, con el fin de facilitar la mantenibilidad.

3.2.3 Requisitos de hardware y software

Para seleccionar la herramienta de simulación/emulación de red más adecuada para el desarrollo de los escenarios experimentales, se definieron varios requisitos que debe cumplir el simulador. En la Tabla 8 se evalúan Mininet, GNS3, EVE-NG, NS-3 y OMNeT++ según dichos requisitos, marcando con "X" cuando el simulador los satisface.

Tabla 8. *Requerimientos de simuladores de red*

Nº	Requisito	Mininet	GNS3	EVE-NG	NS-3	OMNeT++
RS1	Soporte de conmutadores virtuales (vSwitch) y hosts	X	X	X	X	X
RS2	Soporte para trabajar con protocolo OpenFlow (controladores SDN)	X	X	X	X	X
RS3	Instalación sencilla en cualquier equipo	X	X	X		
RS4	Facilidad para crear y modificar topologías de red para distintos escenarios de prueba		X	X		X

RS5	Soporte multiplataforma para ejecución en entornos de laboratorio (principalmente Linux/Windows)	X	X	X	X	X
RS6	Adecuado desempeño y escalabilidad para los escenarios definidos en la tesis	X	X	X	X	X

De igual forma a los simuladores, se definieron requisitos que debe cumplir el controlador SDN para soportar la implementación y evaluación de los algoritmos de balanceo de carga Round Robin y LSTM. En la Tabla 9 se comparan POX, Ryu, OpenDaylight (ODL) y ONOS, marcando con "X" cuando el controlador cumple el requisito.

Tabla 9. Requisitos del Controlador

Nº	**Requisito**	**POX**	**Ryu**	**ODL**	**ONOS**
RC1	Implementado en un lenguaje que facilite la integración con librerías de aprendizaje profundo (Python)	X	X		
RC2	Facilidad de uso y curva de aprendizaje adecuada para entornos académicos y de laboratorio	X	X		
RC3	Arquitectura modular que permita desarrollar y probar aplicaciones SDN personalizadas		X	X	X
RC4	Buen desempeño para escenarios de pequeña y mediana escala (latencia, jitter, pérdida aceptables)	X	X	X	X
RC5	Soporte para las versiones de OpenFlow	X	X	X	X
RC6	Documentación y ejemplos suficientes para el desarrollo de aplicaciones de balanceo de carga	X	X	X	X

De acuerdo con la evaluación de los requerimientos técnicos y funcionales, se determina que la herramienta de emulación más adecuada para el desarrollo del presente trabajo es Mininet. Esta elección se justifica por su capacidad para desplegar topologías de red escalables y reproducibles mediante virtualización ligera a nivel de sistema operativo (namespaces), lo que permite un prototipado rápido de escenarios SDN con un

consumo de recursos significativamente menor a la virtualización tradicional. En lo referente al plano de control, se selecciona el controlador Ryu, debido a su arquitectura modular basada en componentes y su implementación nativa en el lenguaje Python. Esta compatibilidad lingüística es crítica, ya que favorece la integración directa de las bibliotecas de aprendizaje profundo (necesarias para el modelo LSTM) y la lógica del algoritmo Round Robin dentro del mismo entorno de ejecución, facilitando además la recolección granular de métricas de tráfico indispensables para el análisis comparativo.

En función de las herramientas seleccionadas (Mininet, Ryu y librerías de Deep Learning para el modelo LSTM), se establecen los requisitos de hardware mínimos y recomendados. Si bien Mininet optimiza el uso de recursos mediante virtualización ligera, la fase de entrenamiento de la red neuronal y el procesamiento de datasets requieren capacidad de cómputo adicional. A continuación, se detallan las especificaciones para el equipo de desarrollo:

Tabla 10. *Requerimientos para correr Mininet*

Componente	**Especificación Recomendada**	**Motivo Principal**
Procesador (CPU)	4 a 8 Núcleos (Intel i5/i7 o Ryzen 5/7)	Paralelismo requerido para el entrenamiento del modelo LSTM.
Memoria RAM	8 GB (Mínimo) – 16 GB (Óptimo)	Carga de datasets en memoria y ejecución concurrente de Ryu.
Almacenamiento	128 GB o superior (**SSD**)	Velocidad de lectura/escritura crítica para logs y trazas.
Sistema Operativo	Linux (Ubuntu 20.04/22.04 LTS)	Ejecución nativa de Mininet sobre el kernel del sistema.

3.3 Diseño del sistema

El diseño del sistema propuesto se fundamenta en una arquitectura de red definida por software (SDN) modular, donde el plano de control se desacopla del plano de datos para permitir una gestión centralizada y programable del tráfico. El sistema integra un módulo de inteligencia artificial basado en Deep Learning para la toma de decisiones de balanceo de carga, superando las limitaciones de los algoritmos estáticos tradicionales.

3.3.1 Arquitectura general SDN

La arquitectura del sistema se estructura en tres planos funcionales, comunicados mediante interfaces estándar:

1. **Plano de Infraestructura (Datos):** Compuesto por conmutadores virtuales (Open vSwitch) que gestionan el reenvío de paquetes basándose en tablas de flujo. Estos dispositivos carecen de inteligencia de control propia y dependen de instrucciones externas.

2. **Plano de Control:** Implementado mediante el controlador **Ryu**, un marco de desarrollo basado en componentes y desarrollado en Python. Ryu actúa como el "cerebro" de la red, comunicándose con el plano de datos a través del protocolo OpenFlow 1.3. Este plano aloja la lógica de enrutamiento, gestión ARP y, crucialmente, el módulo de inferencia de la red neuronal.

3. **Plano de Aplicación:** En este nivel reside la lógica de negocio, específicamente los algoritmos de balanceo de carga (Round Robin y LSTM). Estas aplicaciones interactúan con el controlador mediante APIs internas para modificar el comportamiento de la red en tiempo real según las predicciones de carga.

3.3.2 Diseño de topología de red

Para el desarrollo experimental se opta por una arquitectura de red centralizada basada en el modelo de "Conmutador Único" (*Single-Switch Topology*). La justificación técnica para seleccionar esta topología, radica en la necesidad de aislar el rendimiento del algoritmo de balanceo de carga de otras variables de red. Al eliminar la complejidad de múltiples saltos (*multi-hop*) y protocolos de enrutamiento dinámico, se garantiza que las métricas de latencia, *throughput* y uso de CPU sean resultado directo de la eficiencia del algoritmo de distribución seleccionado y no de la congestión en enlaces intermedios.

Como se muestra en la **Figura 9**, la estructura de la red se organiza en una topología de estrella donde el plano de datos converge en un único punto central de gestión. Este diseño permite que el controlador tenga una visión global de todos los flujos que ingresan y salen del sistema, facilitando la toma de decisiones en tiempo real para la asignación de servidores.

La arquitectura lógica, detallada en la Figura 9, se compone de tres segmentos:

1. **Plano de Control:** Gestionado por el controlador SDN, el cual supervisa el estado de la red e inyecta las reglas de flujo.
2. **Nodo Central de Conmutación:** Un switch compatible con OpenFlow que actúa como intermediario, ejecutando las acciones de reenvío y modificación de cabeceras (NAT) dictadas por el controlador.
3. **Host Finales:** Se dividen en un grupo de clientes, encargados de la inyección de tráfico, y un clúster de cuatro servidores que procesan las peticiones.

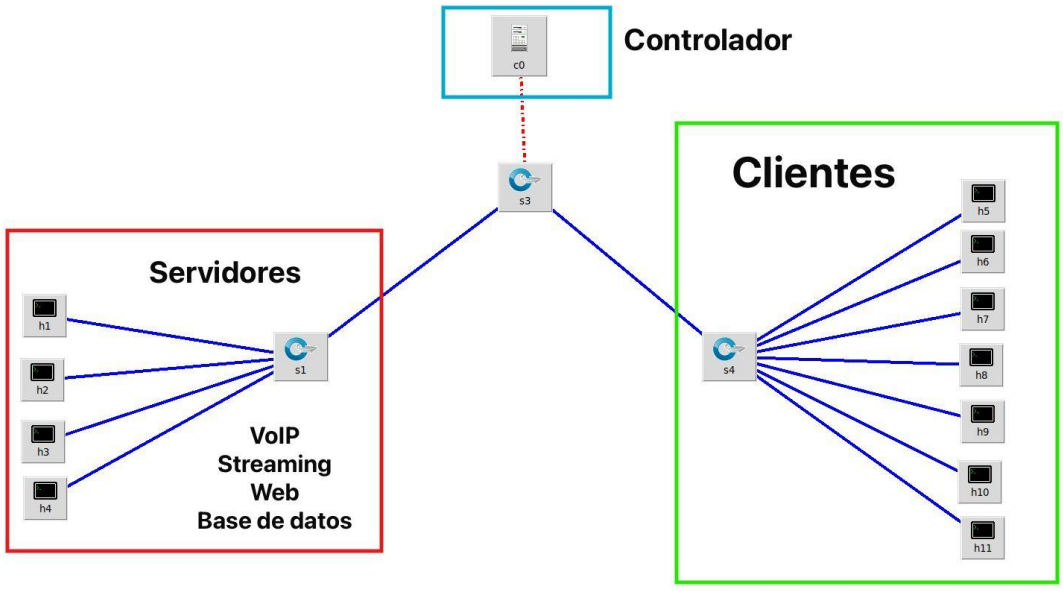
Adicionalmente, el diseño implementa un esquema de direccionamiento basado en IPs Virtuales (VIP) para desacoplar el servicio de la infraestructura física. Esto permite

exponer diferentes servicios en direcciones lógicas específicas, como se describe a continuación:

- **Servicio Web:** Mapeado a la IP Virtual (VIP) 10.0.0.100.
- **Base de Datos:** Mapeado a la VIP 10.0.0.101
- **Streaming:** Mapeado a la VIP 10.0.0.200
- **VoIP:** Mapeado a la VIP 10.0.0.201

Este diseño asegura que, independientemente del servidor físico que resuelva la petición, el cliente siempre interactúe con la interfaz virtual mostrada en el esquema.

Figura 9. *Topología de Red experimental SDN*



3.3.3 Diseño del algoritmo Round Robin

Como línea base para la evaluación comparativa, se desarrolló un módulo de balanceo de carga basado en la política *Round Robin* (RR). La implementación no se limita a la distribución cíclica simple, sino que integra un mecanismo de Persistencia de

Sesión (*Sticky Sessions*) para garantizar la estabilidad de las conexiones orientadas a conexión (TCP) dentro del entorno SDN.

El núcleo del algoritmo se basa en un enfoque determinista de asignación de recursos. Ante la llegada de una nueva solicitud de servicio, el controlador selecciona el servidor destino S_i siguiendo la ecuación $i = (k \ mod \ n)$ en donde:

- k: representa el contador global de solicitudes acumuladas.
- n: es la cardinalidad del conjunto de servidores disponibles (en este escenario, $n = 3$).
- mod: es la operación módulo que asegura la rotación cíclica entre los índices de los servidores $[0,1 \dots, n-1]$

Dado que aplicaciones como HTTP y bases de datos requieren mantener un estado de conexión, una distribución paquete por paquete rompería la comunicación al enviar segmentos de un mismo flujo TCP a servidores diferentes. Para mitigar esto, se implementó una tabla de hash en la memoria del controlador.

El algoritmo opera mediante la identificación de flujos basada en la tupla $T_{id} = \{IP_{origen}, P_{origen}\}$ el flujo lógico del controlador ante un evento Packet-In es el siguiente:

1. **Extracción de Cabeceras:** Se analiza el paquete entrante para obtener T_{id}
2. **Consulta de Estado:** Se verifica si T_{id} ya existe en la tabla de sesiones activas.
3. **Toma de Decisión:**

- Caso A (Sesión Existente): Se omite el cálculo cíclico y se reenvía el paquete al mismo servidor asignado previamente, garantizando la continuidad de la sesión.
- Caso B (Nueva Sesión): Se ejecuta la función Round Robin, se incrementa el contador k, y se registra la nueva asociación $T_{id} -> Servidor_{asignado}$ en la tabla.

Para sostener el esquema de IPs Virtuales descrito en la sección anterior, el algoritmo incorpora un módulo de Proxy ARP. Dado que las VIPs no están asociadas a ninguna interfaz física, el controlador intercepta las solicitudes ARP (ARP_REQUEST) que buscan la dirección MAC de la VIP.

El controlador responde fabricando un paquete ARP_REPLY que asocia la IP Virtual con una MAC Virtual (e.g., 00:00:00:00:FE:80 para Web). Esto fuerza a los clientes a encapsular su tráfico hacia la entidad virtual, permitiendo al controlador interceptarlo posteriormente.

Una vez determinado el servidor destino, el controlador aplica técnicas de modificación de cabeceras en tiempo real (Packet Header Rewriting) mediante OpenFlow:

1. **NAT (Downstream):** En los paquetes que van del Cliente al Servidor, se reemplaza la IP destino (VIP) y la MAC destino (Virtual) por las direcciones físicas reales del servidor seleccionado.
2. **Reverse NAT (Upstream):** En la respuesta del Servidor al Cliente, el controlador intercepta el tráfico y restaura la IP origen (cambiando la IP Real del servidor por la VIP). Esto es crítico para que el cliente acepte la respuesta, ya que espera recibirla de la IP con la que inició la comunicación.

3.3.4 Diseño del modelo neuronal (LSTM)

Para el núcleo de toma de decisiones del sistema, se diseñó una arquitectura de aprendizaje profundo basada en Redes Neuronales Recurrentes de Memoria a Corto y Largo Plazo (LSTM). Específicamente, se optó por una variante Bidireccional (Bi-LSTM), la cual procesa la secuencia de datos tanto en orden cronológico (pasado a futuro) como inverso. Esta topología permite al modelo capturar patrones de congestión complejos que una red unidireccional podría pasar por alto, mejorando la precisión en la predicción del servidor óptimo.

La arquitectura propuesta consta de un flujo secuencial de tensores estructurado en cuatro bloques principales como se muestra en la Figura 10 y se describe a continuación:

1. **Capa de Entrada y Secuenciación:** El modelo recibe como entrada una matriz tridimensional de tamaño (B, T, F) donde:

- B (Batch Size): Es el número de muestras procesadas simultáneamente.
- T (Time Steps): La ventana de tiempo histórica que el modelo "mira" hacia atrás.
- F (Features): El número de características normalizadas (uso de CPU, RAM, ancho de banda, *jitter*, etc.).

2. **Bloque Recurrente (Hidden Layers):** El núcleo de procesamiento consta de 3 capas apiladas de celdas LSTM.

- **Configuración:** 64 unidades ocultas (*hidden units*) por capa.
- **Bidireccionalidad:** Al ser bidireccional, la salida efectiva de cada capa se duplica (64 * 2 = 128 características latentes), concatenando los estados ocultos de las pasadas hacia adelante y hacia atrás.

- **Justificación:** El uso de múltiples capas permite al modelo abstraer características de alto nivel, separando el ruido (fluctuaciones momentáneas de la red) de las tendencias reales de carga.

3. **Mecanismos de Regularización y Estabilidad:** Para garantizar la convergencia del entrenamiento y evitar el sobreajuste (*overfitting*), se integraron dos técnicas fundamentales entre las capas recurrentes:

- **Batch Normalization:** Normaliza la salida de las capas ocultas para mantener la media cercana a 0 y la varianza cercana a 1. Esto acelera el entrenamiento y reduce la sensibilidad a la inicialización de los pesos.

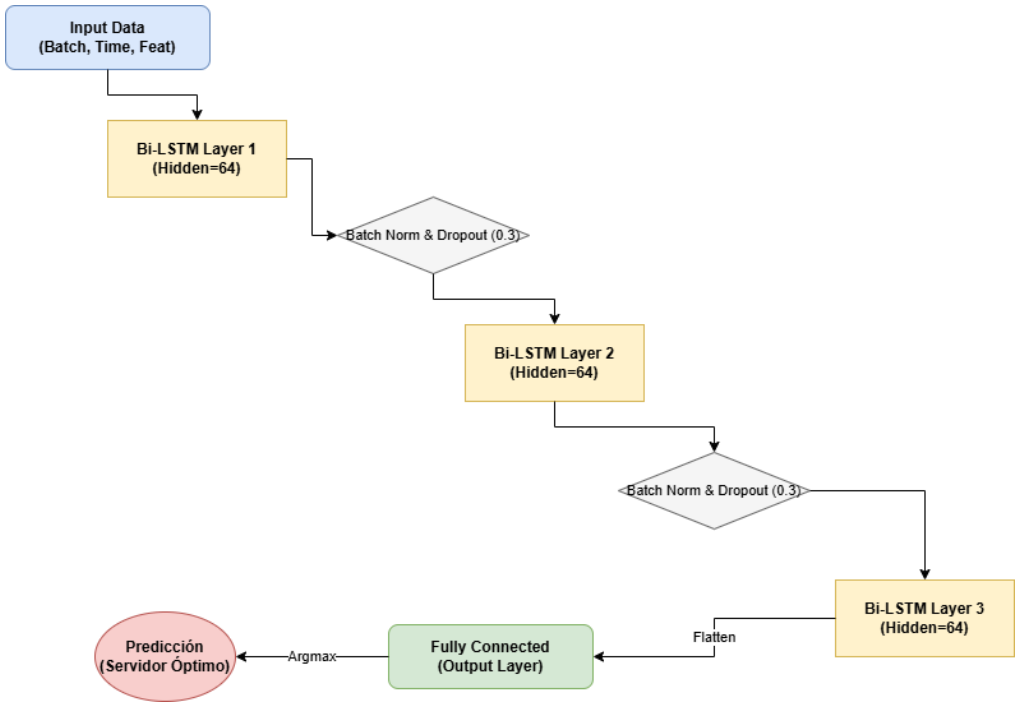
- **Dropout:** Se aplicó una tasa de desconexión del 30% ($p = 0.3$) Durante el entrenamiento, el modelo desactiva aleatoriamente el 30% de las neuronas, forzando a la red a aprender representaciones robustas y redundantes, evitando que dependa excesivamente de un solo nodo.

4. Capa de Salida (Fully Connected): La etapa final es una capa lineal completamente conectada (*Dense Layer*) que transforma el vector de características latentes en un vector de probabilidad.

- **Dimensión de Salida:** Corresponde al número de servidores disponibles en el clúster ($n = 3$).

- **Función de Activación:** El resultado se procesa, implícitamente mediante la función de pérdida, para determinar la clase (servidor) con la mayor probabilidad de ser el TARGET_IDEAL.

Figura 10. *Diagrama de Bloques de LSTM*



La eficacia del modelo no solo depende de su arquitectura, sino de la correcta sintonización de sus hiperparámetros. A continuación en la Tabla 11, se detallan los valores seleccionados para el proceso de optimización estocástica:

Tabla 11. *Parámetros para procesamiento*

Parámetro	Valor	Descripción Técnica
Optimizador	Adam	Algoritmo de optimización adaptativa de momento.
Learning Rate	0.001	Tasa de aprendizaje inicial para el descenso de gradiente.
Función de Pérdida	MSE / CrossEntropy	Métrica para evaluar la discrepancia entre la predicción y el *Target Ideal*.
Batch Size	64	Número de muestras propagadas antes de actualizar los pesos.
Epochs	50 - 100	Ciclos completos de entrenamiento sobre todo el dataset.
Hidden Layers	3	Profundidad de la red recurrente.
Hidden Units	64	Capacidad de memoria de cada celda LSTM.

3.3.5 Creación de Dataset

La eficacia de un modelo de aprendizaje profundo (Deep Learning) depende directamente de la calidad y variabilidad de los datos utilizados durante su entrenamiento. Dado que no existen conjuntos de datos públicos que correlacionen métricas de red SDN con el estado interno de servidores bajo estrés computacional específico, se procedió a la generación de un dataset sintético propio.

La adquisición de datos se automatizó mediante un script desarrollado en Python, diseñado para ejecutarse sobre la topología Mininet descrita en la sección 3.3.2. El proceso de captura tuvo una duración total de 10 horas, operando bajo un esquema Cliente-Servidor sincronizado:

- **Inyección de Tráfico (Plano de Datos):** Los hosts clientes ejecutaron instancias de Iperf para generar flujos de tráfico UDP y TCP variables. Se definieron tres perfiles de carga o "etapas" para garantizar la diversidad del espacio de muestreo:
    - **Etapa BAJA:** Ancho de banda entre 2 y 8 Mbps.
    - **Etapa MEDIA:** Ancho de banda entre 10 y 18 Mbps.
    - **Etapa ALTA:** Ancho de banda entre 20 y 40 Mbps.
- **Agentes de Telemetría (Plano de Servidores):** En cada servidor se desplegó un agente de software residente (Daemon). Este agente cumple una doble función:
    - **Generación de Estrés:** Ejecuta bucles de operaciones aritméticas intensivas para simular carga de procesamiento real, evitando que los servidores permanezcan en reposo (Idle).

- **Monitorización:** Utiliza la librería de sistema psutil para extraer métricas de uso de CPU y memoria RAM en tiempo real, las cuales son reportadas al orquestador central cada segundo.

El resultado del proceso de captura es un registro estructurado donde cada muestra representa el estado global del sistema en un instante (t). Las variables recolectadas se clasifican en:

- **Variables de Red (Input):**
  - **Bandwidth_Solicitado:** Tasa de transferencia demandada por el cliente.
  - **Jitter:** Variación en el retardo de llegada de los paquetes (ms).
  - **Packet_Loss:** Porcentaje de paquetes perdidos en el enlace.
- **Variables de Estado de Servidor (Input):**
  - **CPU_Load:** Porcentaje de uso del procesador.
  - **RAM_Usage:** Porcentaje de ocupación de memoria.

Para entrenar al modelo bajo el paradigma de aprendizaje supervisado, fue necesario establecer una "verdad absoluta" (Ground Truth) que indicara cuál era el servidor óptimo para cada muestra recolectada. Se diseñó una función de coste heurística para calcular el puntaje de capacidades de cada servidor.

Donde (i) representa al servidor evaluado. El sistema etiqueta como TARGET_IDEAL a aquel servidor que minimiza esta función. Esta fórmula prioriza la disponibilidad de CPU como el recurso más crítico, ponderando la memoria RAM con un factor de 0.5 para desempatar escenarios de carga computacional similar.

Debido a que la captura en tiempo real genera un volumen de datos limitado, se aplicó una técnica de aumento de datos (Data Augmentation) para robustecer el modelo y evitar el sobreajuste. El algoritmo de post-procesamiento multiplica cada muestra original por un factor de 20, generando variaciones sintéticas mediante las siguientes técnicas:

- **Perturbación Gaussiana:** Se añade ruido aleatorio a las métricas de CPU y RAM para simular micro-fluctuaciones naturales de los sistemas operativos.
- **Simulación de Redes Hostiles:** Para las muestras clasificadas en la etapa de "Tráfico Alto", el algoritmo inyecta artificialmente condiciones adversas de red:
    - **Jitter Sintético:** Valores aleatorios entre 15ms y 45ms.
    - **Pérdida Sintética:** Tasas de pérdida entre 2% y 8%.

Este proceso incrementó el tamaño del dataset final a aproximadamente 170,000 muestras, proporcionando un orden de datos de entrenamiento suficiente para la convergencia de la red neuronal LSTM.

Una vez consolidada la base de datos ampliada, el aprendizaje profundo exige que la información cruda sea transformada en estructuras matemáticas compatibles. La red neuronal LSTM requiere una escala uniforme en todas sus entradas y una noción dimensional de secuencia.

Para formalizar este proceso de preparación final, se diseñó el un algoritmo que se representa en la Figura 11, el cual representa la lógica secuencial (en formato PSeInt) utilizada para fusionar datos heredados, escalar valores numéricos al rango [0,1] mediante *Min-Max Scaling*, y redimensionar la matriz bidimensional resultante a un tensor tridimensional compatible con la capa de entrada del modelo predictivo.

Figura 11. *Lógica de preprocesamiento, normalización y transformación a tensores.*

```
Algoritmo PreprocesamientoDatasetLSTM

    // Definición de variables para la simulación
    Definir muestras_antiguas, muestras_nuevas, total_muestras Como Entero
    Definir num_caracteristicas Como Entero
    Definir i Como Entero

    num_caracteristicas ← 12
    muestras_antiguas ← 150000
    muestras_nuevas ← 20000

    Escribir "================================================"
    Escribir " PIPELINE ETL: PREPARACIÓN DE DATOS PARA LSTM "
    Escribir "================================================"
    Escribir ""

    // FASE 1: Fusión de Datasets
    Escribir "--- FASE 1: Carga y Fusión de Datos ---"
    Escribir "[+] Cargando dataset_tesis_augmented.csv (3 Servidores)..."
    Escribir "[!] Adaptando estructura: Agregando columnas srv4_cpu=0 y srv4_ram=0."
    Escribir "[+] Cargando dataset_gns3_4srv.csv (4 Servidores)..."

    total_muestras ← muestras_antiguas + muestras_nuevas
    Escribir "[OK] Datasets fusionados. Total de muestras: ", total_muestras
    Escribir ""

    // FASE 2: Separación de Variables
    Escribir "--- FASE 2: Extracción de Características ---"
    Escribir "[+] Extrayendo ", num_caracteristicas, " variables de red y sistema hacia Matriz X."
    Escribir "[+] Extrayendo columna 'TARGET_IDEAL' hacia Vector Y."
    Escribir ""

    // FASE 3: Normalización Min-Max
    Escribir "--- FASE 3: Normalización (Min-Max Scaler) ---"
    Escribir "[+] Comprimiendo valores drásticos (Mbps, ms) al rango estricto [0, 1]..."

    // Simulamos el tiempo de procesamiento de la matriz
    Para i←1 Hasta 3 Con Paso 1 Hacer
        Escribir "    -> Procesando y escalando lote de datos ", i, "..."
    FinPara
    Escribir "[OK] Matriz X normalizada con éxito."
    Escribir ""

    // FASE 4: Codificación One-Hot
    Escribir "--- FASE 4: Codificación Categórica ---"
    Escribir "[+] Transformando etiquetas de texto ('srv1', 'srv2') a valores enteros."
    Escribir "[+] Aplicando One-Hot Encoding (Matriz Binaria Multiclase)."
    Escribir "    Ejemplo lógico: srv1 se convierte en [1, 0, 0, 0]"
    Escribir ""

    // FASE 5: Transformación a Tensores
    Escribir "--- FASE 5: Estructuración Tensorial ---"
    Escribir "[+] Las redes recurrentes requieren la forma: (Muestras, Pasos_Tiempo, Features)"
    Escribir "[+] Redimensionando Matriz bidimensional X a Tensor tridimensional..."
    Escribir "    Forma final de Entrada LSTM: [", total_muestras, ", 1, ", num_caracteristicas, "]
    Escribir ""

    Escribir "================================================"
    Escribir ">>> PROCESO FINALIZADO. DATOS LISTOS PARA IA. <<<"
    Escribir "================================================"
```

3.3.6 Entrenamiento de LSTM

Para la implementación del modelo de red neuronal, se utiliza el framework TensorFlow con la API Keras. Esta elección se fundamenta en su capacidad para la producción y la facilidad para desplegar modelos secuenciales complejos con capas recurrentes.

El script de entrenamiento implementa una estrategia híbrida de ingestión de datos para maximizar la capacidad de generalización del modelo. Se fusionaron dos fuentes de información:

- **Dataset Base:** Datos recolectados de la topología original de 3 servidores.
- **Dataset Extendido:** Un conjunto de datos sintético generado (dataset_gns3_4srv.csv) que simula la presencia de un cuarto servidor para validar la escalabilidad del algoritmo.

Para homogeneizar las entradas, se aplicó una técnica de Imputación de Datos, rellenando las columnas faltantes del dataset antiguo (correspondientes al Servidor 4) con valores sintéticos de "carga infinita" (CPU=100%, RAM=100%), lo que permitió rellenar los vacíos de aprendizaje que podría llegar a tener el modelo. Posteriormente, las variables numéricas (CPU, RAM, Ancho de banda, Jitter) se normalizaron utilizando MinMaxScaler en el rango [0, 1], facilitando la convergencia del descenso de gradiente.

Se diseñó una arquitectura de aprendizaje profundo del tipo Bidireccional, capaz de procesar la secuencia temporal de métricas en ambas direcciones (pasado, futuro y futuro, pasado). La estructura, definida mediante Sequential(), consta de los siguientes bloques:

- **Capa de Entrada:** Recibe vectores de características normalizadas correspondientes a la ventana de tiempo actual.

- **Primera Capa Recurrente:** Bidirectional (LSTM) con 64 unidades de memoria, en donde estas unidades representan la "anchura" de la capa. Es decir, cada paso de tiempo en secuencia de tráfico SDN será transformado en un vector de 64 características aprendidas, además usamos el parámetro return_sequences=True ya que se define como una configuración y su función es permitir que la primera capa LSTM entregue una secuencia completa de datos a la capa siguiente, en lugar de un único valor final. Esto garantiza que la información sobre la evolución temporal del tráfico SDN (latencia, CPU, RAM) no se pierda y pueda ser procesada por las capas superiores para detectar patrones de congestión a largo plazo

- **Segunda Capa Recurrente:** Bidirectional(LSTM) con 32 unidades. Esta capa condensa la información temporal en un vector de características abstracto, es decir, una representación numérica simplificada que contiene los patrones más relevantes del estado de la red. Este vector permite traducir el historial de tráfico en una forma que el modelo puede usar finalmente para clasificar y seleccionar el servidor óptimo en el controlador Ryu.

- **Capas de Regularización:** Entre cada bloque recurrente se intercalan capas de BatchNormalization (para estabilizar la media de las activaciones) y Dropout(0.3) (desactivando aleatoriamente el 30% de las neuronas para evitar el sobreajuste).

- **Capa Densa (Fully Connected):** Una capa oculta de 32 neuronas con activación ReLU debido a su eficiencia computacional y actúa como un umbral de decisión permite que solo las señales relevantes (valores positivos) avancen en la red, descartando la información redundante o ruidosa (valores negativos), lo que facilita que el modelo aprenda relaciones complejas y no lineales entre las métricas de la red SDN.

- **Capa de Salida:** Una capa Dense con activación Softmax. Esta función transforma las salidas de la red en una distribución de probabilidad, donde la suma de todas las salidas es igual a 1. Esto permite al controlador Ryu seleccionar el servidor con la mayor probabilidad de éxito para procesar la carga, garantizando un balanceo basado en la confianza estadística del modelo.

El proceso de optimización se configuró con los siguientes hiperparámetros:

- **Optimizador Adam (Tasa de aprendizaje inicial $\alpha = 0.001$):** Este optimizador fue seleccionado por su capacidad para ajustar dinámicamente la velocidad de aprendizaje de cada parámetro. El valor inicial de 0.001 previene fluctuaciones caóticas al inicio del entrenamiento y establece un descenso estable hacia el error mínimo.
- **Función de Pérdida (categorical_crossentropy):** Es el estándar matemático para la clasificación multiclase. Al operar en sinergia con la capa de salida Softmax, penaliza logarítmicamente las decisiones incorrectas emitidas con alta confianza, forzando al modelo a corregir rápidamente sus probabilidades hacia el servidor óptimo.
- **EarlyStopping (Paciencia = 10):** Actúa como un freno preventivo contra el sobreajuste (*overfitting*). Si la métrica de validación no muestra mejoras durante 10 épocas consecutivas, el entrenamiento se detiene. Esto garantiza que la red conserve su capacidad de generalización y no termine "memorizando" los datos de entrenamiento.
- **ReduceLROnPlateau:** Opera como un mecanismo de precisión. Si el algoritmo de aprendizaje se estanca, es decir, cuando la función de pérdida entra en una meseta matemática donde el gradiente se vuelve insuficiente para generar cambios significativos en los pesos, deteniendo la optimización. Para solventar esto, se

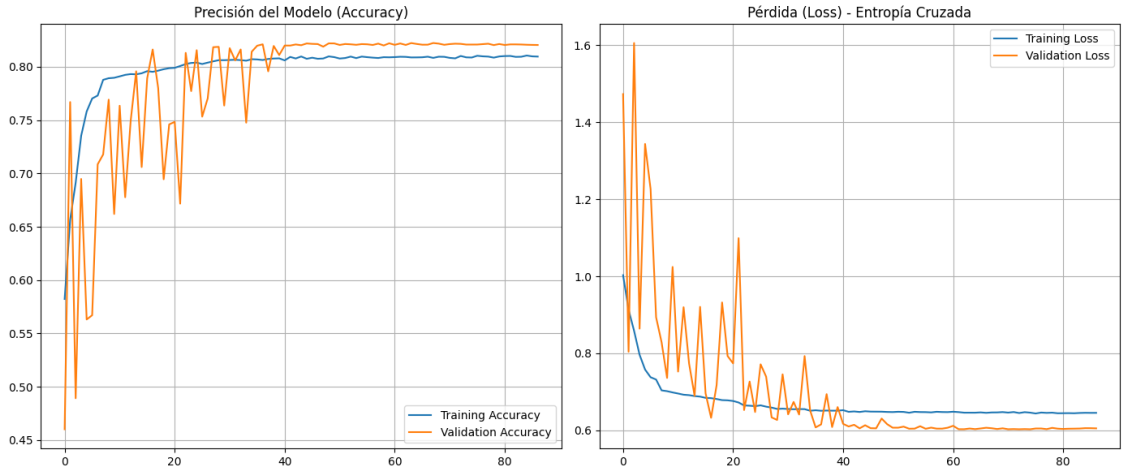
utiliza una función callback, la cual supervisa el entrenamiento en tiempo real. Si tras un número determinado de épocas (patience) no hay mejora, el callback interviene automáticamente reduciendo la tasa de aprendizaje (*Learning Rate*) a la mitad. Este ajuste permite realizar un fine-tuning (ajuste fino), donde el modelo da pasos matemáticos más pequeños y controlados, logrando escapar de mínimos locales y estabilizando la convergencia en los valores óptimos.

El modelo alcanzó una precisión (Accuracy) superior al 82% en el conjunto de prueba, demostrando su capacidad para seleccionar el servidor óptimo.

Para certificar que el modelo alcanzó esta convergencia de manera estable y sin memorizar los datos (*overfitting*), se extrajeron las curvas de aprendizaje durante las 100 épocas de entrenamiento.

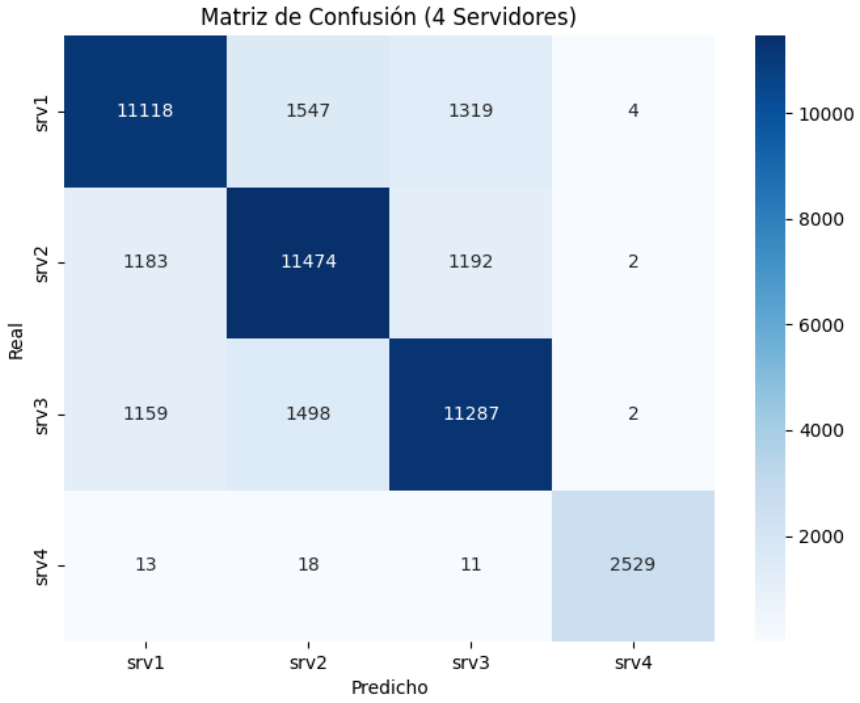
Como se ilustra en la Figura 12, la gráfica de Precisión (Accuracy) muestra un ascenso rápido durante las primeras 20 épocas, estabilizándose por encima de la barrera del 0.80. En paralelo, la gráfica de Pérdida (Loss) demuestra que la curva de validación desciende acompañando estrechamente a la curva de entrenamiento. La ausencia de picos divergentes masivos al final del ciclo confirma la eficacia de las capas Dropout y el callback de EarlyStopping.

Figura 12. *Curvas de precisión y pérdida (Entropía Cruzada) durante el entrenamiento del modelo LSTM.*



Adicionalmente, se evaluó el desempeño clasificatorio del modelo LSTM frente a datos de prueba (no vistos durante el entrenamiento) mediante una Matriz de Confusión (Figura 13). La diagonal principal del gráfico concentra la gran mayoría de las predicciones correctas (Verdaderos Positivos), evidenciando más de 11,000 aciertos individuales para los servidores 1, 2 y 3, y 2,529 aciertos para el servidor 4, proporcionales a la distribución del conjunto de datos. Es destacable que los errores de clasificación (falsos positivos y falsos negativos, ubicados en las áreas claras fuera de la diagonal) mantienen valores marginales en comparación con el volumen total de aciertos.

Figura 13. *Matriz de Confusión evaluando la exactitud predictiva del modelo sobre el clúster de 4 servidores.*



Para sintetizar la secuencia de ejecución desde la definición de la arquitectura hasta la exportación del modelo, se explica en el algoritmo de la Figura 14. Este pseudocódigo ilustra el flujo de control implementado en Python, permitiendo replicar la lógica de aprendizaje profundo independientemente de las librerías subyacentes.

Figura 14. *Ejecución lógica del flujo de entrenamiento y validación de la red neuronal LSTM.*

```
Algoritmo EntrenamientoModeloLSTM

    Definir epoca, max_epocas Como Entero
    Definir loss, accuracy Como Real

    max_epocas ← 100

    Escribir "======================================================"
    Escribir " INICIO DE ENTRENAMIENTO DE RED NEURONAL (LSTM) "
    Escribir "======================================================"
    Escribir ""

    // FASE 1: Construcción de la Arquitectura
    Escribir "--- FASE 1: Definiendo Arquitectura del Modelo ---"
    Escribir "[+] Capa 1: Bidirectional(LSTM) - 64 Unidades (Secuencial)"
    Escribir "[+] Capa 2: BatchNormalization + Dropout(0.3)"
    Escribir "[+] Capa 3: Bidirectional(LSTM) - 32 Unidades (Abstracta)"
    Escribir "[+] Capa 4: Dense Fully Connected - 32 Neuronas (ReLU)"
    Escribir "[+] Capa Salida: Dense - 4 Neuronas (Activacion: Softmax)"
    Escribir ""

    // FASE 2: Compilación
    Escribir "--- FASE 2: Compilando Modelo ---"
    Escribir "[+] Optimizador configurado: ADAM (Learning Rate = 0.001)"
    Escribir "[+] Funcion de Perdida: Categorical Crossentropy"
    Escribir "[+] Callbacks: EarlyStopping(patience=10), ReduceLROnPlateau"
    Escribir ""

    // FASE 3: Bucle de Entrenamiento (Simulación)
    Escribir "--- FASE 3: Ejecutando Entrenamiento (Fit) ---"
    Escribir "Procesando tensores de entrada..."

    // Simulacion de 3 epocas para la consola
    Para epoca ← 1 Hasta 3 Con Paso 1 Hacer
        loss ← 1.5 / epoca
        accuracy ← 0.60 + (0.08 * epoca)
        Escribir "Epoch ", epoca, "/", max_epocas, " -> loss: ", loss, " - accuracy: ", accuracy, " - val_loss: ", (loss+0.1)
    FinPara
    Escribir "... (Entrenamiento continua optimizando pesos) ..."
    Escribir "Epoch 45/", max_epocas, " -> loss: 0.6401 - accuracy: 0.8250 - val_loss: 0.6510"
    Escribir "[!] EarlyStopping ejecutado: La perdida de validacion se estabilizo."
    Escribir ""

    // FASE 4: Validacion y Exportacion
    Escribir "--- FASE 4: Evaluacion y Guardado ---"
    Escribir "[+] Generando Curvas de Aprendizaje (Graficas)..."
    Escribir "[+] Calculando Matriz de Confusion sobre datos de prueba..."
    Escribir "[+] Accuracy final validado: > 82.00%"
    Escribir "[+] Guardando pesos sinapticos en: 'modelo_gns3.h5'"
    Escribir ""
    Escribir "======================================================"
    Escribir ">>> ENTRENAMIENTO FINALIZADO EXITOSAMENTE <<<"
    Escribir "======================================================"

FinAlgoritmo
```

3.4 Implementación

En esta sección se detalla el procedimiento técnico ejecutado para la materialización operativa del escenario experimental. La implementación abarca desde la configuración del entorno de virtualización y las dependencias de software (Mininet, Ryu, TensorFlow) hasta la codificación de los scripts de orquestación para la topología de red y la integración de los algoritmos de balanceo de carga (Round

Robin y LSTM). Asimismo, se describen los parámetros de control de tráfico aplicados a los enlaces para simular restricciones físicas y el protocolo de validación funcional ejecutado para certificar la interoperabilidad entre el plano de datos y el plano de control antes de la fase de pruebas experimentales.

3.4.1 Configuración del entorno experimental (Mininet, Ryu, OpenFlow 1.3)

La implementación del escenario experimental usa la integración de dos ecosistemas de desarrollo distintos: uno orientado a la emulación de redes (Networking) y otro al análisis de datos y aprendizaje profundo (Data Science). A continuación, se detalla el conjunto de librerías seleccionadas y su función específica en la arquitectura de la red y de la arquitectura

Para la definición de la infraestructura de red, no se utilizaron interfaces gráficas, sino que se optó por la definición de topologías mediante código (*Infrastructure as Code*). Se empleó la API nativa de Python para Mininet, la cual permite instanciar nodos y enlaces como objetos de software.

La librería crítica para el realismo de la simulación fue mininet.link, específicamente la clase TCLink. Esta clase actúa como una interfaz hacia el subsistema de control de tráfico del kernel de Linux (tc), permitiendo inyectar restricciones físicas en los enlaces virtuales, además en la Tabla 12 se especifican las librerías usadas para el funcionamiento de todo el entorno de simulación.

Tabla 12. *Librerías de Simulación de Red*

Módulo / Librería	Clase Utilizada	Función Técnica en la Implementación
mininet.net	Mininet	Clase constructora principal que encapsula el objeto de red y gestiona el ciclo de vida de los hosts y switches.
mininet.node	RemoteController	Permite la conexión del plano de datos local hacia un controlador SDN externo (Ryu) ejecutándose en el puerto 6633.
mininet.link	TCLink	Habilita la configuración de parámetros de calidad de servicio (QoS) como ancho de banda (bw), retardo (delay) y pérdida de paquetes (loss).
mininet.log	setLogLevel	Gestiona la granularidad de los logs del sistema para depurar eventos de conexión en tiempo real.

El diseño del modelo neuronal LSTM se sustentó en el ecosistema de librerías científicas de Python. El flujo de trabajo se estructuró en tres etapas: ingestión de datos, preprocesamiento y construcción del modelo.

Para la manipulación de los grandes volúmenes de datos generados durante las pruebas de carga, se utilizó la librería Pandas, permitiendo la fusión de datasets heterogéneos y la limpieza de valores nulos. Posteriormente, Scikit-learn fue fundamental para la normalización, transformando las métricas de red crudas (con rangos muy dispares) a una escala uniforme [0, 1] mediante MinMaxScaler, paso indispensable para la convergencia de la red neuronal.

Finalmente, la arquitectura profunda se implementó sobre TensorFlow/Keras, aprovechando sus capas de alto nivel para definir la estructura recurrente bidireccional. A continuación, en la Tabla 13 se resume las librerías y sus funciones en los controladores y procesamiento de datos.

Tabla 13. *Librerías de Ciencia de Datos e Inteligencia Artificial*

Librería	Módulo Específico	Aplicación en el Modelo
Pandas	DataFrame, read_csv	Estructuración tabular de los logs de tráfico y etiquetado de muestras.
Scikit-learn	MinMaxScaler	Escalamiento de características (CPU, RAM, Bandwidth) para optimizar el descenso de gradiente.
	LabelEncoder	Codificación de las variables categóricas (ID del servidor objetivo) a valores numéricos.
TensorFlow	Sequential	Contenedor lineal para el apilamiento de capas neuronales.
	Bidirectional, LSTM	Implementación de la memoria recurrente capaz de analizar secuencias temporales en ambas direcciones.
Seaborn	heatmap	Generación de matrices de confusión para evaluar visualmente los errores de clasificación del modelo.

3.4.2 Construcción de la topología simulada

Para la validación de los algoritmos de balanceo de carga, se ha diseñado una infraestructura de red bajo el paradigma SDN utilizando la herramienta de emulación Mininet. La topología sigue un diseño jerárquico que permite aislar el tráfico de los clientes del segmento de servidores, interconectados a través de un núcleo de conmutación (Core).

En el desarrollo del script de automatización *topo_full.py*, se han definido variables que representan los recursos de hardware y red virtualizados. La Tabla 14 resume la función de cada una de ellas:

Tabla 14. *Definición de variables del entorno de simulación.*

Variable	Clase / Tipo	Descripción Técnica
net	Mininet	Objeto principal que encapsula la red virtual y gestiona los eventos del ciclo de vida (inicio/parada).
s1, s3, s4	OVSKernelSwitch	Switches Open vSwitch operando bajo el estándar OpenFlow 1.3 para la gestión de flujos.
c0	RemoteController	Instancia que apunta al controlador externo (Ryu), ubicado en la IP 127.0.0.1 puerto 6633.
srv[1-4]	Host	Nodos destinados a la prestación de servicios (WEB, Streaming, BD, VoIP) con IPs 10.0.0.1 a 10.0.0.4.
cli[1-7]	Host	Nodos clientes que actúan como generadores de carga concurrentes en la red.

En términos generales, el script utiliza clases para la definición de nodos (switches, hosts y controladores) y funciones procedimentales para automatizar el despliegue de servidores web, bases de datos y tráfico multimedia en cada uno de los servidores finales. Esta separación de responsabilidades permite que la topología no sea solo un conjunto estático de nodos, sino un entorno dinámico capaz de responder a peticiones reales de red.

A continuación, se describen los componentes funcionales del script:

- **Clase RemoteController**: Esta clase permite que el plano de datos (Mininet) se comunique con un plano de control externo. En este proyecto, se configura para apuntar a la dirección 127.0.0.1, permitiendo que el controlador Ryu tome el control de las tablas de flujo de los switches.
- **Clase OVSKernelSwitch**: Implementa el switch virtual Open vSwitch dentro del kernel de Linux. Se utiliza específicamente para garantizar el soporte del protocolo OpenFlow 1.3, necesario para las operaciones de modificación de cabeceras (NAT) que realizan los algoritmos de balanceo.
- **Función start_service(host, name, port)**: Es la unidad lógica encargada de la capa de aplicación. Realiza tres acciones críticas: crea un entorno de archivos aislado en /tmp para el host, genera una respuesta HTML personalizada que identifica al servidor y levanta un servicio de escucha en un puerto específico (8080 para Streaming, 3306 para DB, 5060 para VoIP). Esto permite diferenciar el tráfico según el servicio solicitado por los clientes.
- **Función start_web(host)**: Una variante especializada de la función de servicio, optimizada para el protocolo HTTP en el puerto estándar 80. Su propósito es

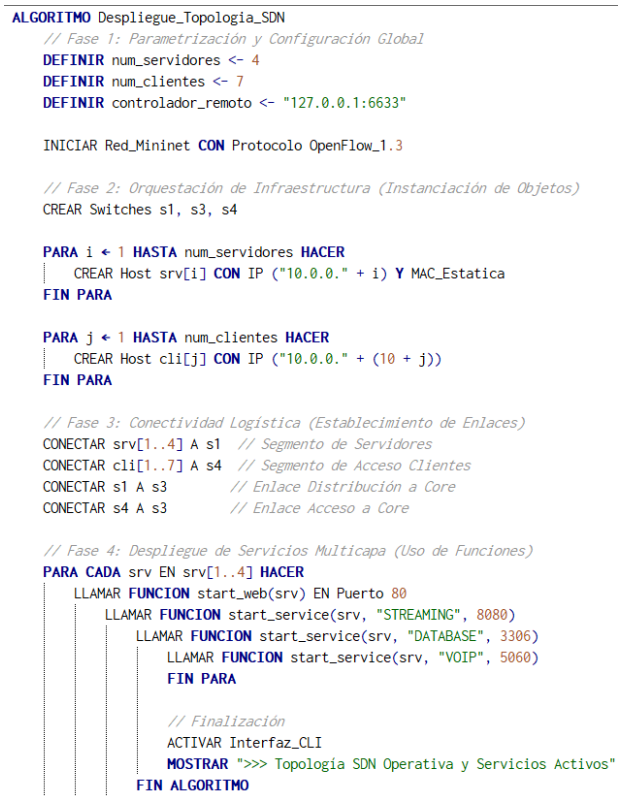
simular la carga de trabajo de un servidor web convencional, facilitando las pruebas de conectividad inicial y el balanceo de peticiones básicas.

- **Función run()**: Actúa como el orquestador principal del experimento. Su lógica comprende desde la instanciación de la red y la creación de enlaces físicos entre los segmentos de clientes y servidores, hasta la activación de los demonios de servicio en cada host y la apertura de la interfaz de comandos para el monitoreo en tiempo real.

Para garantizar la integridad operativa de la infraestructura SDN, el proceso de construcción sigue una lógica secuencial estructurada. Esta abstracción algorítmica permite validar el orden de encendido de los dispositivos y la correcta asociación de servicios antes de la fase de experimentación.

Como se ilustra en la Figura 15, el flujo lógico se divide en procesos de inicialización, interconexión y activación de servicios.

Figura 15. *Algoritmo de despliegue de infraestructura SDN*



```
ALGORITMO Despliegue_Topologia_SDN
    // Fase 1: Parametrización y Configuración Global
    DEFINIR num_servidores <- 4
    DEFINIR num_clientes <- 7
    DEFINIR controlador_remoto <- "127.0.0.1:6633"

    INICIAR Red_Mininet CON Protocolo OpenFlow_1.3

    // Fase 2: Orquestación de Infraestructura (Instanciación de Objetos)
    CREAR Switches s1, s3, s4

    PARA i ← 1 HASTA num_servidores HACER
        CREAR Host srv[i] CON IP ("10.0.0." + i) Y MAC_Estatica
    FIN PARA

    PARA j ← 1 HASTA num_clientes HACER
        CREAR Host cli[j] CON IP ("10.0.0." + (10 + j))
    FIN PARA

    // Fase 3: Conectividad Logística (Establecimiento de Enlaces)
    CONECTAR srv[1..4] A s1   // Segmento de Servidores
    CONECTAR cli[1..7] A s4   // Segmento de Acceso Clientes
    CONECTAR s1 A s3          // Enlace Distribución a Core
    CONECTAR s4 A s3          // Enlace Acceso a Core

    // Fase 4: Despliegue de Servicios Multicapa (Uso de Funciones)
    PARA CADA srv EN srv[1..4] HACER
        LLAMAR FUNCION start_web(srv) EN Puerto 80
            LLAMAR FUNCION start_service(srv, "STREAMING", 8080)
                LLAMAR FUNCION start_service(srv, "DATABASE", 3306)
                    LLAMAR FUNCION start_service(srv, "VOIP", 5060)
                    FIN PARA

                    // Finalización
                    ACTIVAR Interfaz_CLI
                    MOSTRAR ">>> Topología SDN Operativa y Servicios Activos"
                FIN ALGORITMO
```

El proceso inicia con la Fase de Parametrización, donde se inicializan las variables **num_servidores** y **num_clientes**, asignando los valores de 4 y 7 respectivamente para definir la escala del centro de datos. Acto seguido, se declara el uso de un controlador remoto operando en la dirección local para gestionar el plano de control externo.

Posteriormente, se ejecuta la Fase de Orquestación de Infraestructura. En este paso, el algoritmo instancia los objetos **s1, s3 y s4**, que representan los switches Open vSwitch. Luego, se procede a la creación de los nodos finales (**srv1..4** y **cli1..7**). A cada variable de tipo servidor se le asigna una dirección IP estática y una dirección MAC única para su identificación en el controlador, mientras que las variables de cliente se direccionan secuencialmente.

La Fase de Conectividad Logística define la topología jerárquica. El algoritmo utiliza funciones de enlace para vincular los servidores al switch **s1** y los clientes al switch **s4**. La interconexión crítica entre estos segmentos se realiza a través de la variable **s3**, que actúa como el núcleo de la red, asegurando la visibilidad de todo el tráfico bajo el protocolo OpenFlow 1.3.

Finalmente, el algoritmo realiza el Despliegue de Servicios Multicapa. Mediante el uso de las funciones **start_web** y **start_service**, se activan de forma asíncrona cuatro demonios de escucha en cada nodo servidor, diferenciados por las variables de puerto:

- **Puerto 80**: Servicio **WEB** para peticiones HTTP.
- **Puerto 8080**: Servicio **STREAMING** para flujos de video.
- **Puerto 3306**: Servicio **DATABASE** para tráfico de bases de datos.
- **Puerto 5060**: Servicio **VOIP** para señalización de voz sobre IP.

3.4.3 Implementación de controlador Round Robin

El primer controlador desarrollado se basa en el algoritmo Round Robin (RR). Este controlador ha sido programado en Python utilizando el framework Ryu y opera bajo el estándar OpenFlow 1.3. Su objetivo principal es garantizar una distribución equitativa de las cargas en términos de cantidad de conexiones, sin considerar el estado interno de los recursos (CPU o RAM) de los servidores.

A continuación en la Tabla 15, se describen los elementos técnicos fundamentales que componen el script ryu_service_rr.py:

Tabla 15. *Variables y funciones principales del Controlador Round Robin*

Elemento	Tipo	Descripción Técnica
rr_counter	Variable	Contador entero utilizado para indexar y rotar de forma cíclica la selección entre los servidores activos.
SERVICES	Variable (Dict)	Diccionario que mapea las Direcciones IP Virtuales (VIPs) a sus respectivos servicios (Web, Streaming, DB, VoIP) y puertos.
sticky_sessions	Variable (Dict)	Estructura de datos que almacena la persistencia de sesión. Asocia un cliente y un servicio a un servidor específico para evitar la ruptura de flujos TCP.
get_round_robin_server()	Función	Implementa la lógica central del algoritmo. Retorna el identificador del servidor siguiente en la lista y actualiza el valor de rr_counter.
_packet_in_handler()	Función	Punto de entrada del evento PacketIn. Analiza las cabeceras de los paquetes entrantes para decidir si requieren una respuesta ARP o un proceso de balanceo IP.
_balancear()	Función	Ejecuta la modificación de cabeceras (*Destination NAT*). Cambia la IP de destino (VIP) por la IP real del servidor elegido por el algoritmo RR.
_reverse_nat()	Función	Realiza la traducción inversa de direcciones para el tráfico de salida (Server a Cliente), asegurando que el host reciba paquetes con la IP de la VIP original.

Para validar la secuencia de toma de decisiones del balanceador estático, se presenta la Figura 16 con el pseudocódigo. Este describe el procesamiento de paquetes desde su llegada al switch hasta la aplicación de las reglas de flujo para la redirección de tráfico.

Figura 16. *Algoritmo de balanceo de carga Round Robin con gestión de identidades virtuales y persistencia de sesión.*

```
ALGORITMO Controlador_Round_Robin
    // Fase 1: Inicialización de estructuras y constantes virtuales
    DEFINIR rr_counter <- 0
    DEFINIR sticky_sessions <- DICCIONARIO_VACIO

    // Definición de Servicios (VIPs y MACs Virtuales) en la variable SERVICES
    DEFINIR SERVICES <- {
    "10.0.0.100": { "nombre": "WEB",       "vmac": "00:00:00:00:FE:80", "puerto": 80 },
    "10.0.0.101": { "nombre": "STREAMING", "vmac": "00:00:00:00:FE:81", "puerto": 8080 },
    "10.0.0.102": { "nombre": "DATABASE",  "vmac": "00:00:00:00:FE:82", "puerto": 3306 },
    "10.0.0.103": { "nombre": "VOIP",      "vmac": "00:00:00:00:FE:83", "puerto": 5060 }
}
// Fase 2: Gestión de eventos mediante _packet_in_handler()
AL RECIBIR Evento_PacketIn(paquete) HACER:

    // Resolución de ARP para IPs Virtuales
    SI paquete ES TIPO_ARP ENTONCES
        SI IP_solicitada EXISTE EN SERVICES ENTONCES
            RESPONDER ARP con VMAC correspondiente en SERVICES
        FIN SI
        RETORNAR
    FIN SI

    SI paquete ES TIPO_IPV4 ENTONCES
        Extraer IP_Origen, IP_Destino (VIP), Puerto_Destino

        // Fase 3: Persistencia de flujo (Sticky Sessions)
        DEFINIR session_key <- (IP_Origen, Puerto_Destino)

        SI session_key EXISTE EN sticky_sessions ENTONCES
            servidor_elegido ← sticky_sessions[session_key]
        SINO
            // Fase 4: El Proceso Round Robin (Núcleo del Algoritmo)
            // Se invoca la función para decidir el servidor destino
            servidor_elegido ← LLAMAR FUNCION get_round_robin_server()

                ALMACENAR servidor_elegido EN sticky_sessions[session_key]
            FIN SI

            // Fase 5: Aplicación de NAT mediante funciones _balancear() y _reverse_nat()
            SI paquete ES HACIA_SERVIDOR ENTONCES
                LLAMAR FUNCION _balancear(servidor_elegido)
                SINO SI paquete ES DESDE_SERVIDOR ENTONCES
                        LLAMAR FUNCION _reverse_nat()
                        FIN SI
                    FIN SI
                FIN AL RECIBIR

                // --- DEFINICIÓN DE LA FUNCIÓN PRINCIPAL DE SELECCIÓN ---
                FUNCION get_round_robin_server()
                    lista_servidores ← ["srv1", "srv2", "srv3", "srv4"]

                    // Selección basada en el índice actual del contador
                    servidor_seleccionado ← lista_servidores[rr_counter]

                    // Actualización cíclica del rr_counter
                    rr_counter ← (rr_counter + 1) MODULO 4

                    RETORNAR servidor_seleccionado
                FIN FUNCION
            FIN ALGORITMO
```

Como se detalla en el algoritmo, el componente más crítico es el Proceso Round Robin ejecutado a través de la función get_round_robin_server(). Esta función permite que el sistema no favorezca a ningún nodo físico de manera arbitraria. Cada vez que se detecta una nueva sesión que no existe en sticky_sessions, se consulta el valor actual de rr_counter para indexar la lista de servidores. Inmediatamente después

de la asignación, la función incrementa el contador y aplica una operación de módulo 4, lo que permite que el puntero regrese al primer servidor tras haber recorrido los cuatro nodos disponibles.

El manejo de las identidades virtuales se integra perfectamente en este flujo. La variable SERVICES proporciona las Direcciones IP Virtuales (VIPs) y las MACs Virtuales (VMACs) necesarias para que la función _packet_in_handler() responda a las solicitudes ARP y clasifique el tráfico IPv4.

Finalmente, una vez que get_round_robin_server() entrega el servidor destino, las funciones _balancear() y _reverse_nat() actúan como los mecanismos de ejecución técnica, modificando las cabeceras de los paquetes en tiempo real para sustituir las direcciones virtuales por las físicas de los servidores srv1 a srv4, manteniendo así la transparencia del balanceo de carga frente al usuario final.

3.4.4 Implementación de controlador RNN-LSTM

El segundo controlador desarrollado integra un modelo de Redes Neuronales Recurrentes (RNN), específicamente del tipo Long Short-Term Memory (LSTM), para gestionar el balanceo de carga de manera predictiva y dinámica. A diferencia del enfoque estático de Round Robin, este controlador actúa como un agente inteligente que evalúa el estado de recursos de los servidores (CPU y RAM) y métricas de red antes de asignar una solicitud.

La implementación utiliza el framework Ryu en conjunto con la librería TensorFlow/Keras para el uso del modelo. El controlador carga un "cerebro" previamente entrenado *(modelo_gns3.h5)* y utiliza escaladores de datos para normalizar las métricas de telemetría, permitiendo que la red SDN se adapte proactivamente a las ráfagas de tráfico y evite la saturación de nodos específicos.

En la Tabla 16 se detallan las variables y funciones que permiten la integración de la IA en el plano de control:

Tabla 16. *Variables y funciones del sistema de balanceo inteligente LSTM*

Elemento	Tipo	Descripción Técnica
self.model	Objeto (Keras)	Almacena el modelo LSTM cargado desde el archivo .h5. Es el encargado de realizar la predicción del servidor óptimo.
self.scaler	Objeto (Joblib)	Escalador de datos utilizado para normalizar las métricas de entrada (0 a 1), asegurando la compatibilidad con los pesos de la red neuronal.
METRICS_FILE	Variable (String)	Ruta del archivo JSON que contiene la telemetría en tiempo real (CPU, RAM de cada servidor) recolectada por los agentes externos.
sticky_sessions	Variable (Dict)	Diccionario de persistencia que garantiza que, una vez que la IA elige un servidor para un flujo, este se mantenga durante toda la sesión.
predict_best_server()	Función	Extrae las últimas métricas del JSON, las pre-procesa y ejecuta self.model.predict para obtener el servidor con menor probabilidad de saturación.
_packet_in_handler()	Función	Interceptor de eventos que, ante un nuevo flujo IPv4, solicita una decisión al modelo de IA en lugar de seguir un orden secuencial.
_balancear()	Función	Ejecuta la acción de *Destination NAT* en el switch basándose en la predicción de la red neuronal.
_reverse_nat()	Función	Realiza la traducción inversa para mantener la coherencia de la comunicación entre el servidor elegido por la IA y el cliente.

La implementación del balanceador permite la incorporación de una fase de implementación basada en redes neuronales. En la Figura 17 se presenta el diseño algorítmico del controlador, detallando el flujo de datos desde la captura de métricas hasta la ejecución de la decisión de balanceo en el plano de datos.

Figura 17. *Algoritmo de balanceo con RNN-LSTM*

```
ALGORITMO Controlador_Inteligente_LSTM
    // Fase 1: Inicialización y Carga de Pesos de IA
    CARGAR self.model DESDE "modelo_gns3.h5"     // Modelo de Red Neuronal Entrenado
    CARGAR self.scaler DESDE "scaler_gns3.joblib" // Normalizador de Datos
    DEFINIR sticky_sessions <- DICCIONARIO_VACIO

    // Definición de Servicios con Identidades Virtuales (VIPs y VMACs)
    DEFINIR SERVICES <- {
"10.0.0.100": { "vmac": "00:00:00:00:FE:80", "puerto": 80 },
"10.0.0.101": { "vmac": "00:00:00:00:FE:81", "puerto": 8080 },
"10.0.0.102": { "vmac": "00:00:00:00:FE:82", "puerto": 3306 },
"10.0.0.103": { "vmac": "00:00:00:00:FE:83", "puerto": 5060 }
}
// Fase 2: Gestión de Eventos mediante _packet_in_handler()
AL RECIBIR Evento_PacketIn(paquete) HACER:

    SI paquete ES TIPO_ARP ENTONCES
        RESPONDER con VMAC correspondiente en SERVICES
        RETORNAR
    FIN SI

    SI paquete ES TIPO_IPV4 ENTONCES
        Extraer IP_Origen, IP_Destino (VIP), Puerto_Destino
        DEFINIR session_key <- (IP_Origen, Puerto_Destino)

        // Fase 3: Persistencia y Consulta al Cerebro Artificial
        SI session_key EXISTE EN sticky_sessions ENTONCES
            servidor_elegido ← sticky_sessions[session_key]
        SINO
            // Fase 4: Proceso de Predicción Inteligente
            // Invocación a la lógica de la Red Neuronal
            servidor_elegido ← LLAMAR FUNCION predict_best_server()

                ALMACENAR servidor_elegido EN sticky_sessions[session_key]
            FIN SI

            // Fase 5: Ejecución de NAT con Predicción de IA
            SI paquete ES HACIA_SERVIDOR ENTONCES
                LLAMAR FUNCION _balancear(servidor_elegido)
                SINO SI paquete ES DESDE_SERVIDOR ENTONCES
                        LLAMAR FUNCION _reverse_nat()
                        FIN SI
                    FIN SI
                FIN AL RECIBIR

                // --- NÚCLEO DE INTELIGENCIA ARTIFICIAL ---
                FUNCION predict_best_server()
                    // 1. Adquisición de Telemetría
                    datos_crudos ← LEER_ULTIMA_METRICA(METRICS_FILE) // CPU y RAM de srv1..4

                    // 2. Pre-procesamiento y Normalización
                    // Se usa el escalador para que los datos sean compatibles con el modelo
                    datos_normalizados ← self.scaler.transform(datos_crudos)

                    // 3. Inferencia mediante el Modelo Entrenado
                    // self.model procesa la serie temporal y predice el mejor nodo
                    prediccion_red_neuronal ← self.model.predict(datos_normalizados)

                    // 4. Selección del Servidor Óptimo
                    indice_ganador ← OBTENER_MAXIMA_PROBABILIDAD(prediccion_red_neuronal)

                    RETORNAR "srv" + (indice_ganador + 1)
                FIN FUNCION
            FIN ALGORITMO
```

El funcionamiento del sistema, como se ilustra en el pseudocódigo, se divide en tres etapas críticas coordinadas por el controlador Ryu:

1. **Fase de Preparación y Carga de Inteligencia:** Al iniciar, el controlador utiliza la variable self.model para cargar los pesos del modelo LSTM previamente

entrenado (modelo_gns3.h5). Simultáneamente, se instancia self.scaler, un componente que contiene los parámetros de normalización que transforma los valores de CPU, RAM y Ancho de Banda a un valor decimal que varia en números entre 0 y 1. Esta escala permite que las capas ocultas de la red neuronal ponderen equitativamente el impacto del hardware en el rendimiento de la red. Finalmente, la red neuronal interpreta estos tensores normalizados para predecir qué servidor presenta la menor carga de trabajo, evitando así el sesgo que producirían los valores en bruto.

2. **Proceso de Inferencia Dinámica (Función predict_best_server):** En esta etapa el sistema accede al archivo METRICS_FILE para obtener el estado actual de CPU y RAM de los servidores. Dado que estas métricas tienen magnitudes distintas (porcentajes vs. megabytes), se invoca a self.scaler para normalizar los valores a un rango de [0, 1]. Una vez homogeneizados, los datos son entregados a self.model.predict, función que analiza los patrones de carga y determina qué servidor tiene la menor probabilidad de saturación.

3. **Gestión de Flujos y Persistencia:** Tras obtener la predicción, el controlador consulta la variable sticky_sessions. Si se trata de un flujo nuevo, registra la decisión de la IA para mantener la coherencia de la sesión. Finalmente, las funciones _balancear() y _reverse_nat() ejecutan la modificación de cabeceras en el switch, utilizando las IPs y MACs Virtuales definidas en la variable SERVICES, logrando que la complejidad de la inteligencia artificial sea totalmente transparente para los usuarios finales.

3.5 Técnicas de Análisis de Datos

Para evaluar la eficacia del balanceador de carga basado en RNN-LSTM frente al método Round Robin, se aplica un enfoque cuantitativo y comparativo. El análisis se centra en el comportamiento de los flujos de tráfico IP a través del plano de datos SDN, utilizando métodos estadísticos para procesar la información recolectada en el archivo de métricas. El objetivo es determinar si la inteligencia artificial logra una distribución de carga que minimice la saturación de los recursos del servidor y optimice la transferencia de datos.

3.5.1 Definición de métricas de rendimiento

Siguiendo las recomendaciones de la normativa ITU-T Y.1540, la cual define los parámetros de rendimiento de transferencia de paquetes en redes IP, se han seleccionado métricas permiten medir la calidad de la experiencia del usuario y la eficiencia de la red (International Telecommunication Union, 2023). A continuación, se detallan las métricas aplicadas en esta investigación:

- **IP Packet Transfer Delay (IPTD) - Latencia:** Representa el tiempo transcurrido desde que el paquete ingresa al switch de entrada hasta que es entregado al servidor final. En esta tesis, se analiza cómo el tiempo de inferencia del modelo LSTM impacta en la latencia total frente a la respuesta inmediata del algoritmo Round Robin.

- **IP Packet Loss Ratio (IPLR) - Pérdida de Paquetes:** Es el cociente entre los paquetes totales perdidos y los paquetes enviados en un intervalo de tiempo. Esta métrica es crítica para evaluar si el balanceador inteligente logra evitar la caída de conexiones en escenarios de alta concurrencia mediante el desvío preventivo de flujos.

- **IP Packet Delay Variation (IPDV) - Jitter:** Mide la variación en el tiempo de llegada de los paquetes consecutivos. Es una métrica esencial para los servicios de Streaming y VoIP (puertos 8080 y 5060) configurados en la topología, donde la estabilidad del flujo es prioritaria.

Además, complementando los parámetros de calidad de transferencia, se adopta la metodología de la RFC 2544 (*Benchmarking Methodology for Network Interconnect Devices*). Esta normativa se aplica para determinar el límite de rendimiento del plano de datos gestionado por el controlador SDN, permitiendo cuantificar la eficiencia de la arquitectura bajo condiciones de estrés (Bradner & McQuaid, 1999).

A continuación, se describen las métricas de rendimiento de capacidad medidas bajo este estándar:

- **Throughput (Tasa de transferencia efectiva):** Representa la tasa máxima a la cual el sistema puede procesar y retransmitir paquetes sin pérdida alguna. En la presente investigación, esta métrica permite comparar si el tiempo de procesamiento de la red neuronal (self.model) genera algún cuello de botella en la velocidad de transmisión real en comparación con el algoritmo de selección directa Round Robin.

- **Back-to-Back Frames (Ráfagas de tramas consecutivas):** Mide la capacidad de los buffers de los switches y la velocidad de respuesta del controlador ante ráfagas de tráfico masivo enviadas a la máxima velocidad del medio. Se utiliza para validar la robustez de la lógica de balanceo cuando múltiples clientes solicitan servicios de Streaming y VoIP de manera simultánea.

- **System Recovery (Recuperación del sistema):** Evalúa el tiempo que le toma al controlador SDN volver a un estado estable tras haber sido sometido a una condición de sobrecarga de tráfico. Esta métrica es vital para demostrar que el modelo RNN-LSTM es capaz de estabilizar la red tras una congestión severa.

4 CAPÍTULO IV: PRUEBAS

4.1 Definición de las Pruebas y Escenarios de Carga

Para evaluar el desempeño del modelo predictivo LSTM frente al algoritmo estático Round Robin, se despliega un entorno de emulación en Mininet gestionado por el controlador Ryu (OpenFlow 1.3). La fase experimental se fundamentó en los estándares RFC 2544 e ITU-T Y.1540 para garantizar la validez técnica de las métricas.

Este apartado detalla los parámetros de experimentación en tres fases: primero, la configuración de las ráfagas de tráfico (dimensionamiento de tramas y tiempos de inyección); segundo, el mecanismo de telemetría en el plano de control para capturar el estado de la red (CPU, RAM, ancho de banda) y alimentar el tensor de la IA; y tercero, la definición de los escenarios de prueba. Estos escenarios escalan desde tráfico normal hasta condiciones de saturación y degradación de enlaces, permitiendo medir la latencia, equidad y resiliencia de ambos balanceadores de carga, además se medirá el consumo de CPU y RAM de cada uno de los controladores.

4.1.1 Configuración de las ráfagas de tráfico y parámetros de simulación.

La evaluación experimental se realiza en una topología jerárquica emulada en Mininet, utilizando Open vSwitch y el protocolo OpenFlow 1.3. Siguiendo los estándares RFC 2544 e ITU-T Y.1540, se establecieron los siguientes parámetros técnicos para garantizar un entorno de estrés controlado:

- **Capacidad de Enlaces:** Se configura un enlace troncal (*trunk*) restrictivo de 10 Mbps, mientras que los enlaces de acceso se establecieron en 100 Mbps, se usa estos valores para forzar artificialmente el cuello de botella en la red, ya que al limitar el canal a 10 Mbps, se garantiza que las ráfagas de tráfico superen la capacidad del enlace, permitiendo medir el comportamiento de los algoritmos de balanceo en condiciones críticas de congestión, tal como lo sugiere la metodología de benchmarking del RFC 2544.

- **Gestión de Colas:** Se definie un tamaño de cola (*buffer*) de **50 paquetes** en los switches core para establecer un escenario de "cola corta". En redes SDN, una cola (*buffer*) excesivamente grande ocultaría la pérdida de paquetes a costa de incrementar masivamente la latencia. Al definir una cola de 50, se logra una sensibilidad alta para la métrica IPLR (IP Packet Loss Ratio) de la normativa ITU-T Y.1540. Esto obliga al modelo LSTM a predecir la saturación antes de que el búfer se desborde, evaluando su capacidad preventiva frente a la reactividad de Round Robin.
- **Segmentación de Servicios:** Se seleccionaron cuatro servicios críticos para representar la heterogeneidad de una red real, cada uno con requisitos de QoS distintos según la ITU-T Y.1540:

- **HTTP (Puerto 80):** Tráfico transaccional donde la integridad de los datos es prioritaria.
- **Streaming (Puerto 8080):** Tráfico intensivo en ancho de banda que demanda un *throughput* sostenido.
- **Base de Datos (Puerto 3306):** Tráfico sensible a la latencia (*IPTD*) para evitar el *timeout* de las aplicaciones.
- **VoIP (Puerto 5060):** Tráfico en tiempo real extremadamente sensible al Jitter (*IPDV*). Esta diferenciación permite demostrar que el balanceador inteligente no solo distribuye paquetes, sino que es capaz de priorizar o segregar flujos basándose en el comportamiento predictivo de cada servicio.

- **Dinámica de Carga:** La configuración de un mínimo de **70 paquetes** por flujo responde a dos requisitos:
  - **Significancia Estadística:** Según el RFC 2544, para que una medición de *throughput* o latencia sea válida, debe existir un flujo constante que permita estabilizar los estados de las tablas de flujo (*flow tables*) en los switches.
  - **Ventana Temporal de la IA:** El modelo LSTM requiere de una secuencia histórica de datos (pasos de tiempo) para realizar una predicción precisa. 70 paquetes, distribuidos en el tiempo de la prueba, aseguran que el controlador Ryu recolecte suficientes muestras de telemetría para llenar el tensor de entrada de la red neuronal, permitiendo que esta "aprenda" la tendencia de la carga antes de tomar una decisión de balanceo.

4.1.2 Procedimiento de recolección de telemetría

La recolección de métricas se diseña bajo un esquema de monitoreo híbrido que abarca los planos de datos, control y aplicación. Este procedimiento es el encargado de transformar el estado de la red en tensores de datos procesables por el modelo LSTM. El proceso se divide en tres niveles:

1. **Monitoreo del plano de datos (Red):** Para obtener las métricas de rendimiento exigidas por la ITU-T Y.1540, el sistema ejecuta sondas activas desde los nodos clientes hacia los servidores:
    - **Latencia (IPTD) y Pérdida (IPLR):** Se utilizan ráfagas de mensajes ICMP (*ping*) para calcular el retardo de ida y vuelta y la tasa de descarte de paquetes en tiempo real.
    - **Variación del Retardo (IPDV/Jitter):** Se calcula mediante el análisis estadístico de la diferencia de tiempos de llegada entre paquetes consecutivos.
    - **Ancho de Banda (Throughput):** Se emplea la herramienta iPerf3 en modo cliente-servidor para cuantificar la tasa de transferencia efectiva (Mbps) a través del enlace troncal saturado.
2. **Monitoreo de Host y Plano de Aplicación (Servidores):** Dado que el balanceo de carga depende del estado de salud de los servidores finales, por ello, se recolectan los porcentajes de uso de CPU y RAM de cada servidor mediante sus identificadores de proceso (PIDs) en Mininet. Esto se hace por tres razones técnicas:
    - **Prevención de Cuellos de Botella (Saturación Interna):** Permite detectar cuándo un servidor está saturado a nivel de software, aunque la

red esté libre. La IA usa esto para dejar de enviarle tráfico antes de que empiece a descartar paquetes.

- **Correlación de Carga por Servicio:** Al medir la CPU/RAM, el modelo LSTM "aprende" cuánto le cuesta al servidor procesar cada tipo de tráfico por ejemplo el balanceador analiza que el Streaming exige mucho más procesamiento que una consulta Web y balancea en consecuencia.
- **Medición de Equidad Matemática:** Con estos porcentajes se calcula la Desviación Estándar σ entre los 4 servidores. Si la desviación es cercana a cero, demuestra empíricamente que el balanceador es "justo" y todos los servidores están trabajando por igual.

3. **Monitoreo del Plano de Control (Controlador RYU):** En una red SDN, el "cerebro" (controlador) puede convertirse en un cuello de botella si el algoritmo de balanceo es muy complejo. Por ello, se implementó un monitoreo dedicado al proceso de Ryu para medir su impacto por tres razones clave:
    - **Cuantificación del "Costo" de la IA (Overhead):** Ejecutar una red neuronal como LSTM exige más potencia que un cálculo simple de Round Robin. Medir la CPU y RAM del controlador permite demostrar si la mejora en la red justifica el esfuerzo computacional extra o si el controlador corre el riesgo de saturarse.
    - **Latencia de Decisión (Response Time):** Se mide el tiempo exacto que tarda la IA en "pensar" una decisión de flujo, asimismo se calcula el tiempo exacto de decisión con el controlador Round Robin. Esto es crítico porque, si el controlador tarda demasiado en responder a un paquete nuevo, se genera un retraso (lag) en toda la red, afectando la experiencia del usuario.

- **Garantía de Escalabilidad:** Este monitoreo permite validar que el sistema es eficiente. Al registrar cómo varían los recursos del controlador según la carga de tráfico, podemos asegurar que la solución es capaz de manejar redes más grandes sin colapsar el plano de control.

4.1.3 Diseño de los Casos de Prueba

La evaluación experimental se estructura en diez escenarios diseñados para someter al sistema de balanceo a condiciones de red crecientes y anómalas. Siguiendo el rigor del RFC 2544 para el *benchmarking* de dispositivos de red y la ITU-T Y.1540 para la precisión de transferencia de paquetes IP, las pruebas transicionan desde un estado de operación basal hasta eventos de saturación crítica y degradación forzada. Esta metodología permite cuantificar la capacidad de respuesta, la equidad en la distribución de recursos y la resiliencia tanto del algoritmo Round Robin como del modelo LSTM. A continuación, se detallará la Tabla 17 en la cual se muestra el resumen de la configuración, métricas, escenarios y objetivo de cada una de las pruebas realizadas:

Tabla 17. *Resumen de pruebas realizadas*

Prueba	Denominación	Carga y Escenario	Métrica Primaria	Objetivo Técnico
1	Tráfico Base	Baja (70-100 pps). Peticiones secuenciales.	IPTD (Latencia base)	Establecer la línea base del sistema en estado ideal.
2	Operación Diaria	Media (100-200 pps). Concurrencia de 12 hosts.	Equidad (CPU/RAM)	Medir la estabilidad de la distribución en condiciones normales.
3	Estrés Crítico	Alta. iPerf masivo > 10 Mbps (Trunk saturado).	IPLR (Pérdida)	Forzar el desbordamiento de búferes para medir recuperación.

4	Tráfico Pesado	Alta TCP sostenida en puerto 8080.	Throughput Total	Evaluar el rendimiento máximo en servicios de streaming.
5	Ráfagas (Spikes)	Variable. Transición súbita de 10 a 300 pps.	Tiempo de Reacción	Medir la velocidad de adaptación de la IA ante cambios bruscos.
6	Degradación (Jitter)	70 pps. Retardo de 50ms forzado en srv1 y srv2.	IPDV (Jitter)	Validar el desvío preventivo de flujos hacia rutas estables.
7	Congestión Core	Alta. 5% de pérdida forzada en switches core.	IPLR / Buffers	Evaluar la predicción de pérdida en el plano de datos.
8	Tráfico Mixto	Media-Alta. 50% HTTP y 50% Streaming.	Equidad de Carga	Comprobar la resiliencia ante flujos heterogéneos.
9	Fallo de Nodo	Media. Saturación extrema (100% CPU) en srv4.	Failover / Resiliencia	Validar la detección y exclusión automática de nodos fallidos.
10	Estabilidad	Fluctuante (70-300 pps) por 15 minutos.	Estabilidad Temporal	Garantizar la robustez del sistema en periodos prolongados.

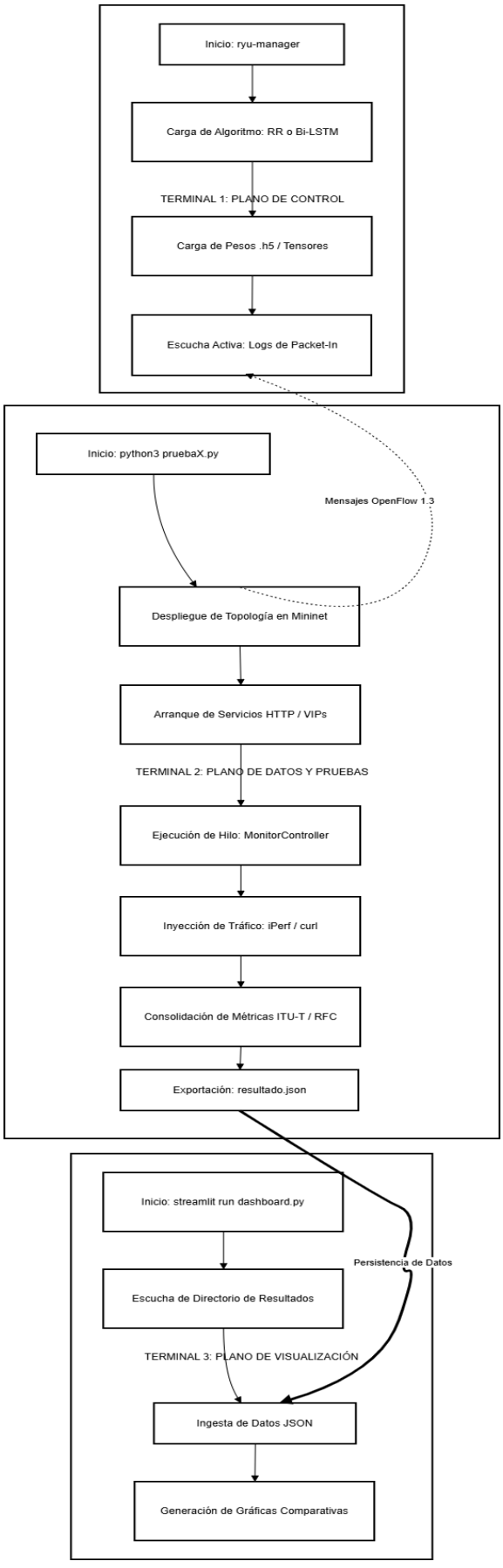
Para garantizar un análisis comparativo justo y reproducible, el sistema opera bajo un flujo de trabajo cíclico y automatizado presentado en la Figura 18. Este proceso integra el plano de control (Ryu), el plano de datos emulado (Mininet), la lógica de experimentación (Scripts de prueba) y la capa de visualización (Dashboard).

El flujo de trabajo se descompone en cuatro fases principales:

1. **Fase de Control:** Inicialización del controlador Ryu cargando el algoritmo correspondiente (estático o inteligente).
2. **Fase de Ejecución:** Orquestación de la topología y ejecución de la prueba seleccionada vía terminal, inyectando el tráfico y aplicando restricciones de red.

3. **Fase de Persistencia:** Recolección de telemetría y consolidación de resultados en archivos estructurados JSON.
4. **Fase de Análisis:** Procesamiento de los datos para su representación gráfica en un panel de control interactivo.

Figura 18. *Flujo de ejecución de pruebas*



Una vez definida la dinámica de ejecución multi-terminal, es necesario detallar la configuración técnica y lógica de cada escenario experimental. Esta sección desglosa las diez pruebas diseñadas, vinculando las constantes programadas en los *scripts* de Python con los objetivos de estrés y medición. Cada escenario está estructurado para validar una faceta específica de la red SDN, permitiendo observar la transición desde un estado de operación ideal hasta condiciones críticas de saturación y degradación.

A continuación, se presentan los algoritmos de prueba, iniciando con el establecimiento de la línea base del sistema.

1. **Prueba 1: Trafico Base.** Este escenario constituye el punto de referencia. Su propósito es medir el rendimiento de la red en condiciones ideales, sin congestión. Los resultados permiten identificar el retardo intrínseco (IPTD) que añade el procesamiento del controlador y el modelo LSTM en estado de reposo. La configuración de la prueba se rige por la Tabla 18 en donde se establecen las variables y parámetros que se usan para obtener los resultados que se han establecidos

Tabla 18. *Variables y parametros de la prueba 1*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total de ejecución	30 segundos
MIN_PACKETS	Volumen de peticiones por flujo	70 paquetes
VIP_WEB	Dirección del servicio objetivo	10.0.0.100 (Puerto 80)
BW_TRUNK	Capacidad del enlace troncal	10 Mbps
QUEUE_SIZE	Límite del búfer del switch	50 paquetes
SEED	Semilla de reproducibilidad	42
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba es completamente automatizada para garantizar la precisión de las mediciones temporales. El algoritmo presentado en la Figura 19 inicia desplegando la topología y los servicios emulados. Luego, inyecta tráfico de forma secuencial y espaciada (con

retardos de 0.3 segundos) para garantizar que el canal troncal no supere su capacidad máxima, evitando cualquier descarte de paquetes. Finalmente, detiene la prueba y consolida las métricas basales de red (IPTD) y de consumo de recursos en un archivo estructurado.

Figura 19. *Algoritmo de la Prueba 1 (Tráfico Base)*

```
ALGORITMO Prueba1_TraficoBase
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba1.json"

    INICIO
    // Fase 1: Inicialización del Entorno
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico Controlado
    PARA iteracion = 1 HASTA MIN_PACKETS HACER
        PARA CADA cliente EN [h1, h2, h3, h4] HACER
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB")
            Esperar(0.3 segundos) // Retardo para evitar ráfagas Back-to-Back
        FIN PARA
    FIN PARA

    // Fase 3: Captura de Métricas (ITU-T y RFC 2544)
    latencia_iptd ← Ejecutar_Sondas_ICMP()
    metricas_servidores ← Capturar_CPU_RAM_Procesos()
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    // Fase 4: Persistencia de Datos
    Estructura_Resultados ← {
"latencia_base": latencia_iptd,
"consumo_nodos": metricas_servidores,
"overhead_controlador": tiempo_respuesta_ryu,
"perdida_paquetes": 0%  // Validación de estado ideal
    }

    Guardar_Archivo_JSON("resultado_prueba1.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

2. **Prueba 2: Operación Normal.** Este escenario emula la operación diaria de la red bajo una carga transaccional moderada y concurrente. Su propósito es evaluar la capacidad del algoritmo (Round Robin o LSTM) para distribuir equitativamente el tráfico entre los servidores en un entorno multiusuario, evitando la sobrecarga individual de los nodos. Los resultados permiten cuantificar la equidad y balanceo mediante la desviación estándar del uso de CPU y RAM en el clúster. La configuración

de la prueba se rige por la Tabla 19, en donde se establecen las variables y parámetros que se usan para obtener los resultados que se han establecido.

Tabla 19. *Variables y parámetros de la prueba 2*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total de ejecución	60 segundos
CLIENTES_CONCURRENTES	Nodos inyectando tráfico simultáneo	12 hosts
VIP_WEB	Dirección del servicio objetivo	10.0.0.100 (Puerto 80)
INTERVALO_MUESTREO	Frecuencia de captura de CPU/RAM	1 segundo
BW_TRUNK	Capacidad del enlace troncal	10 Mbps
QUEUE_SIZE	Límite del búfer del switch	50 paquetes
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba es completamente automatizada para emular el comportamiento de múltiples usuarios accediendo al servicio simultáneamente. El algoritmo presentado en la Figura 20 inicia desplegando la topología y los servicios emulados. A diferencia de la prueba base, inyecta tráfico de forma paralela desde 12 clientes, forzando al controlador a tomar decisiones continuas de enrutamiento. Durante la inyección, captura muestras periódicas del consumo de recursos para medir la estabilidad del balanceo. Finalmente, detiene la prueba y consolida las métricas de equidad (σ) y rendimiento en un archivo estructurado.

***Figura 20**. Algoritmo de la Prueba 2 (Operación Diaria)*

```
ALGORITMO Prueba2_OperacionDiaria
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba2.json"

    INICIO
    // Fase 1: Inicialización del Entorno
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico Concurrente
    MIENTRAS tiempo_actual < DURACION (60s) HACER
        PARA CADA cliente EN [cli1, cli2, ..., cli12] EN PARALELO HACER
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB")
        FIN PARA

        // Fase 3: Captura Constante de Métricas de Equidad
        SI (tiempo_actual MOD 1_segundo == 0) ENTONCES
            metricas_actuales ← Capturar_CPU_RAM_Procesos()
            Agregar_A_Historial(metricas_actuales)
        FIN SI
        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    desviacion_cpu, desviacion_ram <- Calcular_Desviacion_Estandar(Historial)
    throughput_agregado ← Medir_Throughput_Servidores()
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"equidad_cpu_std": desviacion_cpu,
"equidad_ram_std": desviacion_ram,
"throughput_total": throughput_agregado,
"overhead_controlador": tiempo_respuesta_ryu
    }

    Guardar_Archivo_JSON("resultado_prueba2.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

3. **Prueba 3: Estrés Crítico.** Este escenario simula un evento de congestión severa en la red, inyectando un volumen de tráfico que supera intencionalmente la capacidad máxima del canal troncal. Su propósito principal es forzar el desbordamiento de los búferes en los conmutadores OpenFlow para medir la Tasa de Pérdida de Paquetes IP (IPLR), una de las métricas más críticas de la normativa ITU-T Y.1540. Los resultados permiten comprobar si el modelo LSTM es capaz de predecir la saturación inminente y redirigir los flujos antes de que ocurra el descarte masivo, comparado con la respuesta reactiva del algoritmo Round Robin. La

configuración de la prueba se rige por la Tabla 20, en donde se establecen las variables y parámetros que se usan para obtener los resultados de estrés.

Tabla 20. *Variables y parámetros de la prueba 3*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total de inyección de estrés	60 segundos
TRAFICO_MASIVO	Herramienta de saturación de canal	iPerf3 (TCP sostenido)
BW_TRUNK	Capacidad limitante del enlace	10 Mbps (oversubscription)
QUEUE_SIZE	Límite estricto del búfer	50 paquetes
METRICA_OBJETIVO	Variable principal a evaluar	IPLR (Pérdida de paquetes)
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada se centra en saturar el enlace de 10 Mbps. El algoritmo presentado en la Figura 21 despliega la topología y lanza herramientas de generación de tráfico masivo (iPerf3) para colapsar las colas de 50 paquetes. De forma paralela, inyecta flujos de control para contabilizar exactamente cuántos paquetes son descartados por la red. Finalmente, consolida la métrica de IPLR junto con el consumo de recursos de la IA bajo presión, guardando los resultados en un archivo estructurado.

Figura 21. *Algoritmo de la Prueba 3 (Estrés Crítico)*

```
ALGORITMO Prueba3_EstresCritico
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba3.json"

    INICIO
    // Fase 1: Inicialización del Entorno Restrictivo
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servidor_iPerf(Puerto=5201)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico de Saturación
    Iniciar_Cronometro(DURACION = 60s)

    // Inyección masiva para colapsar el enlace troncal
    PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
        Ejecutar_Comando_Consola(cliente, "iperf3 -c VIP_STREAMING -t 60 -b 15M")
    FIN PARA

    // Fase 3: Captura Síncrona de Pérdida de Paquetes (IPLR)
    MIENTRAS Cronometro < DURACION HACER
        total_enviados, total_recibidos <- Ejecutar_Sondas_Diagnostico(Ping_Flood)
        consumo_controlador ← Capturar_CPU_RAM(Ryu)
        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    tasa_iplr_porcentaje ← Calcular_IPLR(total_enviados, total_recibidos)
    throughput_degradado ← Obtener_Resultados_iPerf()
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"perdida_paquetes_iplr": tasa_iplr_porcentaje,
"throughput_sostenido": throughput_degradado,
"cpu_controlador_estres": consumo_controlador,
"overhead_controlador": tiempo_respuesta_ryu
    }

    Guardar_Archivo_JSON("resultado_prueba3.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

4. **Prueba 4: Streaming Pesado.** Este escenario evalúa la capacidad del sistema para gestionar flujos de larga duración y alto volumen de datos característicos de las aplicaciones de transmisión de video (*streaming*). Su propósito es medir el rendimiento máximo (*Throughput*) sostenido que el algoritmo de balanceo puede proporcionar sin colapsar la red. A diferencia del tráfico transaccional (Web), el *streaming* exige un ancho de banda constante y prolongado. Los resultados permiten comprobar si el modelo predictivo LSTM mantiene una asignación eficiente de los recursos a largo plazo frente a la distribución estática de Round Robin. La configuración

de la prueba se rige por la Tabla 21, en donde se establecen las variables y parámetros que se usan para obtener los resultados de rendimiento continuo.

Tabla 21. *Variables y parámetros de la prueba 4*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total de inyección continua	60 segundos
TRAFICO_PESADO	Herramienta de generación de carga	iPerf3 (TCP sostenido)
VIP_STREAMING	Dirección del servicio de video	10.0.0.101 (Puerto 8080)
BW_TRUNK	Capacidad del enlace troncal	10 Mbps
METRICA_OBJETIVO	Variable principal a evaluar	Throughput Agregado (Mbps)
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba es completamente automatizada para garantizar una inyección ininterrumpida de carga. El algoritmo presentado en la Figura 22 inicia desplegando la topología SDN y los servicios correspondientes. Posteriormente, utiliza generadores de tráfico para establecer conexiones TCP persistentes hacia la Dirección IP Virtual de Streaming, intentando consumir todo el ancho de banda disponible del canal troncal. Durante el ciclo de prueba, el sistema monitoriza de cerca las tasas de transferencia efectiva (*Throughput*) en cada nodo. Finalmente, detiene la transmisión, consolida la métrica de rendimiento agregado y exporta los datos de telemetría en un archivo estructurado.

Figura 22. *Algoritmo de la Prueba 4 (Tráfico Pesado - Streaming)*

```
ALGORITMO Prueba4_TraficoPesado
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba4.json"

    INICIO
    // Fase 1: Inicialización del Entorno
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_Streaming(Puerto=8080)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico Sostenido (Elephant Flows)
    Iniciar_Cronometro(DURACION = 60s)

    PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
        // Se solicitan flujos TCP constantes hacia la VIP de Streaming
        Ejecutar_Comando_Consola(cliente, "iperf3 -c VIP_STREAMING -t 60")
    FIN PARA

    // Fase 3: Captura de Rendimiento y Overhead
    MIENTRAS Cronometro < DURACION HACER
        consumo_controlador ← Capturar_CPU_RAM(Ryu)
        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    throughput_individuales ← Extraer_Reportes_iPerf_Servidores()
    throughput_total ← Sumar_Ancho_Banda(throughput_individuales)
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"throughput_agregado_mbps": throughput_total,
"throughput_por_servidor": throughput_individuales,
"overhead_controlador": tiempo_respuesta_ryu,
"cpu_controlador_promedio": Promedio(consumo_controlador)
    }

    Guardar_Archivo_JSON("resultado_prueba4.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

5. **Prueba 5: Ráfagas.** Este escenario evalúa la capacidad de adaptación y el tiempo de reacción del sistema ante cambios abruptos en la demanda de la red (fenómenos conocidos como micro-ráfagas). Su propósito es medir la resiliencia del algoritmo de balanceo cuando el tráfico transiciona violentamente de un estado de reposo a un pico de saturación, y viceversa. Los resultados permiten verificar si el modelo LSTM puede predecir y mitigar la congestión durante la ráfaga de forma más rápida que el enrutamiento estático de Round Robin, minimizando el impacto en la latencia y previniendo la caída de los servidores. La configuración de la

prueba se rige por la Tabla 22, en donde se establecen los umbrales de inyección para forzar la perturbación temporal.

Tabla 22. *Variable y parámetros de la prueba 5*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total del experimento	60 segundos
TRAFICO_BASE	Tasa de peticiones en reposo	10 paquetes por segundo
TRAFICO_PICO	Tasa de peticiones en ráfaga	300 paquetes por segundo
VENTANA_RAFAGA	Momento de activación del pico	Segundos 20 al 40
METRICA_OBJETIVO	Variable principal a evaluar	Tiempo de reacción (ms) y adaptación
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada se divide en tres ventanas temporales continuas (reposo, pico, reposo). El algoritmo presentado en la Figura 23 inicializa la topología y comienza inyectando una carga base muy baja. Repentinamente, multiplica la tasa de peticiones simultáneas, saturando los búferes y los recursos de procesamiento del controlador. Durante esta perturbación, el sistema captura intensivamente el tiempo de respuesta del controlador (*Packet-In Delay*) y las métricas de red. Una vez finalizado el pico, retorna al estado base para medir el tiempo de estabilización antes de consolidar los resultados en el archivo estructurado.

Figura 23. *Algoritmo de la Prueba 5 (Ráfagas Súbitas - Spikes)*

```
ALGORITMO Prueba5_RafagasSubitas
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba5.json"

    INICIO
    // Fase 1: Inicialización del Entorno
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)
    Iniciar_Cronometro(DURACION = 60s)

    // Fase 2: Inyección de Tráfico Escalonado
    MIENTRAS Cronometro < DURACION HACER
        tiempo_actual ← Leer_Cronometro()

        SI (tiempo_actual ≥ 0 Y tiempo_actual < 20) O (tiempo_actual > 40) ENTONCES
            // Tráfico Base (Reposo / Estabilización)
            Tasa_Inyeccion ← TRAFICO_BASE (10 pps)
        SINO
            // Ráfaga Súbita (Spike / Saturación)
            Tasa_Inyeccion ← TRAFICO_PICO (300 pps)
        FIN SI

        PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB", Tasa_Inyeccion)
        FIN PARA

        // Fase 3: Captura Constante de Telemetría
        latencia_ms ← Ejecutar_Sondas_ICMP()
        tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()
        Registrar_Muestra_Temporal(tiempo_actual, latencia_ms, tiempo_respuesta_ryu)

        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    tiempo_reaccion_ia ← Calcular_Tiempo_Adaptacion(Historial_Packet_In_Delay)
    latencia_pico_max ← Extraer_Maximo(Historial_Latencia)

    Estructura_Resultados ← {
"latencia_durante_rafaga": latencia_pico_max,
"tiempo_reaccion_controlador": tiempo_reaccion_ia,
"estabilidad_recuperacion": Calcular_Recuperacion_Post_Rafaga(),
"historial_temporal": Historial_Telemetria
    }

    Guardar_Archivo_JSON("resultado_prueba5.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

6. **Prueba 6: Degradación de Enlace (Jitter)**. Este escenario evalúa la respuesta del sistema ante fluctuaciones de latencia y degradación de la calidad del canal, enfocándose específicamente en la Variación del Retardo de Transferencia de Paquetes IP (IPDV), comúnmente conocida como *Jitter* (normativa ITU-T Y.1540). Su propósito es validar si el modelo predictivo LSTM es capaz de detectar la degradación en rutas específicas (mediante la inyección de un retardo artificial en dos servidores) y desviar preventivamente el tráfico sensible al tiempo (como

la emulación de VoIP) hacia los nodos estables. Por el contrario, se espera que el algoritmo Round Robin continúe enviando tráfico a los nodos degradados de manera "ciega". La configuración de la prueba se rige por la Tabla 23, en donde se establecen las variables y parámetros que se usan para forzar la inestabilidad.

Tabla 23. *Variables y parámetros de la prueba 6*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total de inyección	60 segundos
VIP_VOIP	Dirección del servicio en tiempo real	10.0.0.201 (Puerto 5060)
TRAFICO_BASE	Tasa de peticiones por flujo	70 paquetes
DEGRADACION_FORZADA	Retardo inyectado en enlaces	50ms (en srv1 y srv2)
METRICA_OBJETIVO	Variable principal a evaluar	IPDV (Jitter en ms)
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada altera directamente las propiedades físicas emuladas de los enlaces. El algoritmo presentado en la Figura 24 inicia desplegando la topología, pero aplica comandos de control de tráfico (tc) para sumar 50ms de retardo constante en las interfaces de srv1 y srv2. Una vez degradado el entorno, inyecta el tráfico emulado de VoIP y monitorea constantemente la variación de los tiempos de llegada de los paquetes. Finalmente, detiene la transmisión, consolida la métrica de IPDV y el rendimiento del controlador, guardando los resultados en el archivo estructurado.

Figura 24. *Algoritmo de la Prueba 6 (Degradación - Jitter)*

```
ALGORITMO Prueba6_DegradacionJitter
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba6.json"

    INICIO
    // Fase 1: Inicialización y Degradación del Entorno
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    // Inyección de retardo artificial usando control de tráfico (tc)
    Aplicar_Retardo_Enlace(Nodo="srv1", Retardo="50ms")
    Aplicar_Retardo_Enlace(Nodo="srv2", Retardo="50ms")

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_VoIP_Emulado(Puerto=5060)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico Sensible al Retardo
    Iniciar_Cronometro(DURACION = 60s)

    MIENTRAS Cronometro < DURACION HACER
        PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
            // Envío de tráfico UDP/TCP hacia el puerto de VoIP
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_VOIP")
        FIN PARA

        // Fase 3: Captura Síncrona de Jitter (IPDV)
        variacion_retardo ← Ejecutar_Sondas_Diagnostico_Jitter()
        consumo_controlador ← Capturar_CPU_RAM(Ryu)
        Registrar_Muestra_Temporal(Cronometro, variacion_retardo)

        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    jitter_promedio ← Calcular_Promedio(Historial_Jitter)
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"jitter_ipdv_ms": jitter_promedio,
"nodos_penalizados": ["srv1", "srv2"],
"cpu_controlador": Promedio(consumo_controlador),
"overhead_controlador": tiempo_respuesta_ryu
    }

    Guardar_Archivo_JSON("resultado_prueba6.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

7. **Prueba 7: Congestión Core (Pérdida Forzada).** Este escenario evalúa la resiliencia del sistema frente a fallos parciales en la infraestructura de red, específicamente simulando conmutadores de núcleo (*core switches*) defectuosos o enlaces altamente congestionados que descartan paquetes de forma aleatoria. Su propósito es forzar un incremento severo en la Tasa de Pérdida de Paquetes IP (IPLR) definida por la ITU-T Y.1540, sin necesidad de saturar el ancho de banda. Los resultados permiten verificar si el modelo LSTM tiene la capacidad de identificar la degradación en el plano de datos y mitigar el impacto, en contraste con el enrutamiento estático de Round Robin que no discrimina el estado de los enlaces. La

configuración de la prueba se rige por la Tabla 24, en donde se establecen los parámetros de inyección de fallos artificiales.

Tabla 24. *Variables y parámetros de la prueba 7*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total de inyección	60 segundos
PERDIDA_FORZADA	Tasa de descarte inyectada en enlaces	5% (en switches core)
TRAFICO_BASE	Volumen de peticiones por flujo	70 paquetes
BW_TRUNK	Capacidad del enlace troncal	10 Mbps
METRICA_OBJETIVO	Variable principal a evaluar	IPLR y Retransmisiones
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada altera el comportamiento de la capa de enlace. El algoritmo presentado en la Figura 25 inicializa la topología aplicando políticas de control de tráfico (tc) para forzar un 5% de pérdida de paquetes constante en los enlaces troncales del núcleo. Una vez aplicado el fallo, se inyecta tráfico concurrente desde los clientes y se monitorea intensivamente la métrica de IPLR extremo a extremo, así como el comportamiento de los búferes. Finalmente, se detiene la prueba, se recopilan las estadísticas de retransmisión TCP y el uso de recursos, exportando los datos a su archivo estructurado.

Figura 25. *Algoritmo de la Prueba 7 (Congestión Core)*

```
ALGORITMO Prueba7_CongestionCore
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba7.json"

    INICIO
    // Fase 1: Inicialización e Inyección de Fallos
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    // Inyección de pérdida artificial usando control de tráfico (tc)
    Aplicar_Perdida_Enlace(Enlace="core_links", Tasa_Descarte="5%")

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico bajo Congestión Artificial
    Iniciar_Cronometro(DURACION = 60s)

    MIENTRAS Cronometro < DURACION HACER
        PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
            // Envío de tráfico TCP que forzará retransmisiones al perder paquetes
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB")
        FIN PARA

        // Fase 3: Captura Síncrona de Pérdida Extremo a Extremo
        total_enviados, total_recibidos <- Ejecutar_Sondas_Diagnostico(Ping_Flood)
        consumo_controlador ← Capturar_CPU_RAM(Ryu)

        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    tasa_iplr_efectiva ← Calcular_IPLR(total_enviados, total_recibidos)
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"iplr_medido_porcentaje": tasa_iplr_efectiva,
"descarte_configurado": "5%",
"cpu_controlador_promedio": Promedio(consumo_controlador),
"overhead_controlador": tiempo_respuesta_ryu
    }

    Guardar_Archivo_JSON("resultado_prueba7.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

8. **Prueba 8: Tráfico Mixto.** Este escenario evalúa la robustez y equidad del sistema frente a una demanda heterogénea, combinando flujos de red con diferentes requisitos de Calidad de Servicio (QoS). En un entorno de producción real, el tráfico no es uniforme; por ello, esta prueba inyecta simultáneamente tráfico transaccional ligero (HTTP) y tráfico pesado sostenido (*Streaming*). Su propósito es exponer la principal debilidad del algoritmo estático Round Robin, el cual distribuye conexiones de forma equitativa sin considerar el "peso" computacional de cada flujo. Los

resultados permiten validar si el modelo predictivo LSTM es capaz de correlacionar el tipo de servicio con el consumo de recursos de los servidores, logrando un balanceo justo a pesar de la asimetría del tráfico. La configuración de la prueba se rige por la Tabla 25, en donde se establecen los parámetros de la inyección híbrida.

Tabla 25. *Variables y parámetros de la prueba 8*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total del experimento híbrido	60 segundos
PROPORCION_TRAFICO	Distribución de la inyección de carga	50% Web / 50% Streaming
VIP_WEB	Dirección del servicio transaccional	10.0.0.100 (Puerto 80)
VIP_STREAMING	Dirección del servicio de alto consumo	10.0.0.101 (Puerto 8080)
BW_TRUNK	Capacidad limitante del enlace	10 Mbps
METRICA_OBJETIVO	Variable principal a evaluar	Equidad de Carga (CPU/RAM σ)
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada orquesta un comportamiento dividido en los nodos cliente. El algoritmo presentado en la Figura 26 inicializa la topología y levanta tanto los servicios Web como los de Streaming en el clúster. Posteriormente, divide a los clientes virtuales en dos grupos: el primer grupo inyecta ráfagas intermitentes de peticiones web, mientras que el segundo grupo establece conexiones de alto rendimiento para saturar el canal. Durante esta contienda de flujos, el sistema monitoriza intensivamente la Desviación Estándar (σ) del consumo de CPU y RAM. Finalmente, detiene la emisión y consolida las métricas de equidad y ancho de banda en el archivo estructurado.

Figura 26. *Algoritmo de la Prueba 8 (Tráfico Mixto)*

```
ALGORITMO Prueba8_TraficoMixto
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba8.json"

    INICIO
    // Fase 1: Inicialización de Entorno Multi-Servicio
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
        Iniciar_Servidor_iPerf(Puerto=8080)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico Heterogéneo
    Iniciar_Cronometro(DURACION = 60s)

    MIENTRAS Cronometro < DURACION HACER
        // Grupo 1: Tráfico ligero y transaccional (Web)
        PARA CADA cliente EN [cli1, cli2] EN PARALELO HACER
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB")
        FIN PARA

        // Grupo 2: Tráfico pesado y sostenido (Streaming)
        PARA CADA cliente EN [cli3, cli4] EN PARALELO HACER
            Ejecutar_Comando_Consola(cliente, "iperf3 -c VIP_STREAMING -t 1")
        FIN PARA

        // Fase 3: Captura de Métricas de Equidad Asimétrica
        metricas_actuales ← Capturar_CPU_RAM_Procesos()
        Agregar_A_Historial(metricas_actuales)

        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    desviacion_cpu, desviacion_ram <- Calcular_Desviacion_Estandar(Historial_Metricas)
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"equidad_cpu_std": desviacion_cpu,
"equidad_ram_std": desviacion_ram,
"desbalanceo_detectado": Evaluar_Impacto_Asimetrico(desviacion_cpu),
"overhead_controlador": tiempo_respuesta_ryu
    }

    Guardar_Archivo_JSON("resultado_prueba8.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

9. **Prueba 9. Fallo de Nodo.** Este escenario evalúa la tolerancia a fallos del sistema y su capacidad de recuperación automática frente a la caída crítica de un servidor del clúster. En entornos de alta disponibilidad, es imperativo que el balanceador de carga detecte rápidamente si un nodo deja de responder para excluirlo de la distribución y evitar la interrupción del servicio. El propósito de esta prueba es inyectar un fallo catastrófico (saturación al 100% de CPU o terminación abrupta del proceso) en uno de los servidores a mitad del experimento. Los resultados permiten contrastar la ceguera del algoritmo Round Robin, el cual continuará enviando paquetes al nodo "muerto", generando errores de conexión, frente a la capacidad del modelo predictivo LSTM para aislar la anomalía y redirigir

el tráfico hacia los nodos saludables. La configuración de la prueba se rige por la Tabla 26, en donde se establecen los parámetros del evento de fallo.

Tabla 26. *Variables y parámetros de la prueba 9*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total del experimento	60 segundos
NODO_FALLIDO	Servidor sometido a estrés crítico	srv4
INSTANTE_FALLO	Momento de inyección de la anomalía	Segundo 30
CARGA_CLIENTES	Peticiones concurrentes continuas	100 paquetes por segundo
METRICA_OBJETIVO	Variable principal a evaluar	Errores de Conexión (Tasa de fallo)
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada divide el tiempo de evaluación en dos ventanas: operación normal y contingencia. El algoritmo presentado en la Figura 27 inicializa la topología de forma estándar. Al alcanzar el segundo 30 de la prueba, el script interviene directamente en el espacio de nombres (*namespace*) del servidor objetivo (srv4) ejecutando un comando de estrés absoluto para colapsar sus recursos. Mientras los clientes continúan exigiendo servicio ininterrumpido, el sistema cuantifica la Tasa de Errores HTTP (peticiones fallidas) y el tiempo que tarda la IA en ejecutar la exclusión del nodo (*Failover*). Finalmente, se consolida el impacto del fallo y se exportan los datos.

Figura 27. *Algoritmo de la Prueba 9 (Fallo de Nodo - Failover)*

```
ALGORITMO Prueba9_FalloNodo
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba9.json"

    INICIO
    // Fase 1: Inicialización de Entorno Saludable
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico y Ejecución del Fallo
    Iniciar_Cronometro(DURACION = 60s)

    MIENTRAS Cronometro < DURACION HACER
        tiempo_actual ← Leer_Cronometro()

        // Inyección de contingencia en el segundo 30
        SI tiempo_actual == 30 ENTONCES
            Aplicar_Estres_Critico(Nodo="srv4", CPU="100%", Memoria="100%")
            Registrar_Evento("Caída de srv4 inyectada")
        FIN SI

        PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
            resultado_peticion ← Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB")
            Registrar_Exito_O_Error(resultado_peticion)
        FIN PARA

        // Fase 3: Captura de Telemetría y Detección de Caída
        metricas_actuales ← Capturar_CPU_RAM_Procesos()
        Agregar_A_Historial(metricas_actuales)

        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Cálculos y Persistencia de Datos
    tasa_errores ← Calcular_Porcentaje_Peticiones_Fallidas()
    tiempo_failover ← Calcular_Tiempo_Deteccion_Anomalia()
    tiempo_respuesta_ryu ← Monitorear_Packet_In_Delay()

    Estructura_Resultados ← {
"tasa_errores_porcentaje": tasa_errores,
"tiempo_failover_ms": tiempo_failover,
"comportamiento_nodos_sanos": Evaluar_Carga(srv1, srv2, srv3),
"overhead_controlador": tiempo_respuesta_ryu
    }

    Guardar_Archivo_JSON("resultado_prueba9.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

10. **Prueba 10: Estabilidad a Largo Plazo**. Este escenario evalúa la robustez del sistema y la persistencia del rendimiento bajo un periodo de operación prolongado y dinámico. Su propósito principal es identificar posibles cuellos de botella acumulativos en el controlador Ryu, y verificar si el modelo LSTM degrada su precisión predictiva o incrementa su *overhead* computacional con el paso del tiempo. Los resultados permiten contrastar la simplicidad estructural del algoritmo Round Robin frente a la complejidad de mantener tensores de IA en memoria activa, asegurando que la solución inteligente sea viable para implementaciones en

producción continua. La configuración de la prueba se rige por la Tabla 27, en donde se establecen los parámetros extendidos de evaluación.

Tabla 27. *Variables y parámetros de la prueba 10*

Variable en Código	Descripción	Valor / Parámetro
DURACION	Tiempo total del experimento de fatiga	900 segundos (15 minutos)
PATRON_TRAFICO	Comportamiento de inyección de carga	Fluctuante (Cíclico / Aleatorio)
VIP_WEB	Dirección del servicio objetivo	10.0.0.100 (Puerto 80)
INTERVALO_MUESTREO	Frecuencia de captura de estado	5 segundos
METRICA_OBJETIVO	Variable principal a evaluar	Consumo de RAM (Controlador) y Latencia a largo plazo
MODO	Algoritmo en evaluación	RR / IA

La ejecución de esta prueba automatizada amplía significativamente la ventana de observación frente a los escenarios anteriores. El algoritmo presentado en la Figura 28 inicializa la topología SDN e inyecta un patrón de tráfico fluctuante (que alterna aleatoriamente entre periodos de reposo, tráfico moderado y picos de saturación) durante 15 minutos continuos (900 segundos). Durante este extenso ciclo, el sistema prioriza la recolección exhaustiva del perfil de memoria RAM del controlador y la latencia promedio por minuto, para detectar cualquier tendencia de degradación. Finalmente, detiene la emulación y consolida la línea de tendencia temporal en el archivo estructurado.

Figura 28. *Algoritmo de la Prueba 10 (Estabilidad a Largo Plazo)*

```
ALGORITMO Prueba10_EstabilidadLargoPlazo
ENTRADA: MODO_CONTROLADOR (RR o IA), Topología SDN
SALIDA: Archivo "resultado_prueba10.json"

    INICIO
    // Fase 1: Inicialización de Entorno Prolongado
    Desplegar_Topologia_Mininet(BW_TRUNK=10, QUEUE_SIZE=50)

    PARA CADA servidor EN [srv1, srv2, srv3, srv4] HACER
        Iniciar_Servicio_HTTP(Puerto=80)
    FIN PARA

    Iniciar_Hilo_Monitoreo_Controlador(Ryu)

    // Fase 2: Inyección de Tráfico Fluctuante (Fatiga)
    Iniciar_Cronometro(DURACION = 900s)

    MIENTRAS Cronometro < DURACION HACER
        tiempo_actual ← Leer_Cronometro()
        fase_actual ← Determinar_Fase_Ciclo(tiempo_actual) // Reposo, Moderado o Pico

        Ajustar_Tasa_Inyeccion(fase_actual)

        PARA CADA cliente EN [cli1, cli2, cli3, cli4] EN PARALELO HACER
            Ejecutar_Comando_Consola(cliente, "curl http://VIP_WEB", tasa_actual)
        FIN PARA

        // Fase 3: Muestreo de Salud del Controlador (Memory Leaks)
        SI (tiempo_actual MOD 5_segundos == 0) ENTONCES
            memoria_ryu ← Monitorear_RAM_Proceso(Ryu)
            cpu_ryu ← Monitorear_CPU_Proceso(Ryu)
            latencia_promedio ← Ejecutar_Sondas_ICMP()

            Registrar_Muestra_Temporal(tiempo_actual, memoria_ryu, cpu_ryu, latencia_promedio)
        FIN SI

        Esperar(1 segundo)
    FIN MIENTRAS

    // Fase 4: Análisis de Tendencias y Persistencia
    tendencia_memoria ← Calcular_Pendiente_Crecimiento(Historial_Memoria)
    degradacion_latencia ← Evaluar_Degradacion(Historial_Latencia)

    Estructura_Resultados ← {
"fuga_memoria_detectada": tendencia_memoria > Umbral_Tolerancia,
"consumo_ram_final": Extraer_Ultimo(Historial_Memoria),
"degradacion_rendimiento": degradacion_latencia,
"historial_temporal": Historial_Completo
    }

    Guardar_Archivo_JSON("resultado_prueba10.json", Estructura_Resultados)
    Limpiar_Entorno_Mininet()
FIN
```

La automatización de estos diez escenarios establece un marco de evaluación determinista y reproducible. Al abarcar desde tráfico base hasta fallos de infraestructura y estrés crítico, se valida el comportamiento de la red SDN bajo exigencias reales de producción. Esta metodología garantiza una recolección de métricas exacta, permitiendo una comparación directa y empírica entre el desempeño del balanceo estático (Round Robin) y la eficiencia del modelo predictivo (LSTM).

4.2 **Métricas de Evaluación del Desempeño**

Para cuantificar objetivamente el impacto del modelo neuronal LSTM frente al algoritmo tradicional Round Robin, se establecieron cuatro métricas de rendimiento clave. Estas métricas, alineadas con los estándares de telecomunicaciones ITU-T Y.1540 y la metodología RFC 2544, permiten evaluar tanto la eficiencia de la red en el plano de datos como el consumo computacional en el clúster de servidores. A continuación, se definen los indicadores utilizados para contrastar los resultados de los escenarios de prueba.

4.2.1 Latencia (Tiempo de Respuesta).

En las Redes Definidas por Software (SDN), la latencia es una métrica importante debido a la separación estructural entre el plano de control y el plano de datos (Latif et al., 2020). Cuando un conmutador recibe un paquete que no coincide con ninguna regla de su tabla de flujos, se genera un mensaje *Packet-In* hacia el controlador para solicitar instrucciones, este proceso de consulta y la posterior instalación de flujos mediante mensajes *Flow-Mod* introducen un retardo inevitable en la red (Haji et al., 2021). De acuerdo con la normativa ITU-T Y.1540, este indicador se denomina Retardo de Transferencia de Paquetes IP (IPTD), y su monitoreo es puesto que una carga excesiva en el controlador puede degradar significativamente el tiempo de respuesta y provocar fallos en los servicios de la red (Barrera Pérez & Serrato Losada, 2019; Haji et al., 2021).

La latencia sirve como el principal indicador para cuantificar la sobrecarga computacional de los algoritmos de balanceo. Mientras que métodos tradicionales como Round Robin operan con reglas estáticas de baja complejidad, la integración de modelos de Inteligencia Artificial (IA) requiere procesos de inferencia matricial que

consumen tiempo de CPU (Alhilali & Montazerolghaem, 2023). En este sentido, la métrica permite evaluar si el tiempo de respuesta adicional que introduce el modelo LSTM es aceptable en comparación con la mejora en la eficiencia de distribución de carga que este ofrece frente a controladores convencionales

La latencia se cuantifica en los escenarios de prueba mediante dos enfoques técnicos:

- **Latencia del Plano de Control:** Se extrae directamente de los *logs* del controlador Ryu, midiendo el intervalo entre la llegada de un paquete desconocido y la toma de decisión del algoritmo.
- **Latencia del Plano de Datos (Extremo a Extremo):** Se mide dentro del entorno Mininet utilizando sondas ICMP para calcular el RTT (*Round Trip Time*) y comandos curl para registrar el tiempo de respuesta del servicio HTTP, especialmente en escenarios de tráfico base y ráfagas (Pruebas 1 y 5).

4.2.2 Rendimiento de ancho de Banda (Throughput)

El rendimiento o *throughput* es una métrica fundamental en las Redes Definidas por Software (SDN), especialmente en entornos de centros de datos, ya que cuantifica la tasa efectiva de transferencia de datos exitosa a través de los enlaces de comunicación (Zakia & Ben Yedder, 2017). En las arquitecturas convencionales, el desbalanceo del tráfico y la saturación frecuente de los enlaces suelen degradar drásticamente el rendimiento global. Por lo tanto, medir el ancho de banda efectivo permite determinar la capacidad del sistema para maximizar la utilización de los recursos de red sin provocar cuellos de botella, asegurando que los flujos de datos se transmitan a la mayor velocidad posible permitida por la infraestructura.

El *throughput* actúa como el indicador de eficiencia operativa de los algoritmos de balanceo. Un controlador eficiente debe ser capaz de identificar rutas descongestionadas para mantener una tasa de transferencia alta y constante. Mientras que el algoritmo Round Robin distribuye los flujos de manera ciega, lo que puede llevar a la saturación de enlaces específicos y reducir el rendimiento total, el modelo LSTM tiene un rol proactivo predice la carga futura y selecciona el camino que garantiza el mayor flujo de datos (Alhilali & Montazerolghaem, 2023b). Esta métrica permite validar si la inteligencia añadida al plano de control se traduce en una mejora real de la capacidad de transporte de la red frente a soluciones estáticas.

Para la cuantificación del rendimiento en el entorno emulado, se utiliza un enfoque basado en la saturación controlada de los enlaces:

- **Generación de Carga Sostenida:** Se emplea la herramienta iPerf3 para establecer conexiones TCP persistentes entre los clientes y la IP Virtual (VIP). En el código, esto se gestiona mediante la variable BW_TRUNK, que limita el ancho de banda del enlace troncal a 10 Mbps para forzar la competencia por el recurso.
- **Captura de Datos:** Durante la ejecución (especialmente en la Prueba 4 de *Streaming*), el script registra el volumen de bits transferidos por segundo.
- **Cálculo de Rendimiento Agregado:** Se suma el *throughput* individual de cada servidor para obtener la capacidad total del clúster bajo la gestión del controlador. Estos valores se exportan al archivo JSON para comparar qué tan cerca del límite teórico (10 Mbps) logra operar cada algoritmo.

4.2.3 Tasa de Pérdida de Paquetes (*Packet Loss*).

La tasa de pérdida de paquetes, o *Packet Loss*, es un indicador de la fiabilidad y la calidad del servicio (QoS) en una red. Definida normativamente por la ITU-T Y.1540 como la Relación de Pérdida de Paquetes IP (IPLR), esta métrica cuantifica la fracción de paquetes que, tras ser enviados, no alcanzan su destino debido a congestión en los enlaces o desbordamiento de los búferes en los conmutadores (Zakia & Ben Yedder, 2017). En entornos de centros de datos, la pérdida de paquetes degrada significativamente el rendimiento de las aplicaciones, provocando retransmisiones innecesarias y aumentando la latencia percibida por el usuario final.

Esta métrica permite evaluar la capacidad de prevención de congestión de los controladores evaluados. Mientras que el algoritmo Round Robin distribuye el tráfico de forma cíclica sin considerar el estado de ocupación de las colas en los switches, el modelo LSTM analiza patrones de tráfico para predecir saturaciones inminentes. El rol de la IA es desviar flujos proactivamente hacia rutas alternativas antes de que se alcance el límite de los búferes (50 paquetes en esta arquitectura), minimizando así el descarte de datagramas (Alhilali & Montazerolghaem, 2023b; Wilsonprakash & Deepalakshmi, 2019). Un IPLR menor en el controlador inteligente frente al estático validará la eficacia de la gestión de recursos en escenarios de estrés (Prueba 3) y congestión de núcleo (Prueba 7).

La pérdida de paquetes se calcula mediante un balance comparativo entre el tráfico inyectado y el tráfico procesado:

- **Contabilización de Flujos:** Se utilizan las herramientas iPerf3 y sondas de diagnóstico ping (ICMP) para registrar el número total de paquetes enviados ($P_{enviados}$).

- **Reporte de Recepción:** Se extraen las estadísticas de los servidores de destino para obtener los paquetes que llegaron exitosamente ($P_{recibidos}$).
- **Cálculo Matemático:** Se aplica la fórmula estándar de IPLR: $\boldsymbol{IPLR}(\%) = \left(\frac{\boldsymbol{P_{enviados} - P_{recibidos}}}{\boldsymbol{P_{enviados}}}\right) * \mathbf{100}$

- **Persistencia:** Los resultados porcentuales se almacenan en los archivos JSON, permitiendo identificar en qué niveles de carga (pps) el sistema comienza a degradarse.

4.2.4 Equidad en la Distribución de Carga (Desviación Estándar de CPU/RAM).

En las arquitecturas de centros de datos y redes SDN, el balanceo de carga no se limita únicamente a la gestión del ancho de banda de los enlaces, sino que incluye la distribución eficiente de las peticiones entre los servidores de destino. Cuando las peticiones se distribuyen de manera asimétrica, algunos nodos virtuales pueden sobrecargarse mientras otros permanecen subutilizados (Wilsonprakash & Deepalakshmi, 2019). Los servidores sobrecargados experimentan un agotamiento de sus recursos de procesamiento (CPU) y memoria (RAM), lo que degrada severamente el rendimiento de las aplicaciones alojadas y aumenta el riesgo de caídas del servicio (Zakia & Ben Yedder, 2017) . Para cuantificar matemáticamente la equidad en la asignación de estas cargas, usamos la Desviación Estándar (σ). Esta métrica permite demostrar la "inteligencia" del sistema propuesto. El algoritmo Round Robin distribuye los flujos de forma secuencial y equitativa a nivel de conexiones, pero es "ciego" respecto al costo computacional de cada flujo; una conexión ligera (HTTP) y una pesada (*Streaming*) son tratadas por igual, lo que a menudo desemboca en un

desbalanceo crítico de los recursos (Alhilali & Montazerolghaem, 2023). Por el contrario, el modelo predictivo **LSTM** tiene la capacidad de correlacionar los patrones de red con el estado de los servidores, redirigiendo el tráfico dinámicamente para evitar la saturación individual (Gowri & Baranidharan, 2025). Un valor bajo de desviación estándar en el controlador con IA validará que este logra mantener el clúster en un estado de equilibrio homogéneo, incluso ante inyecciones de tráfico mixto o asimétrico (Prueba 8).

La equidad se mide mediante el monitoreo constante de los procesos emulados en la topología de Mininet:

1. **Muestreo Periódico:** Durante la ejecución de las pruebas, un hilo de monitoreo captura los porcentajes exactos de uso de CPU y RAM de los nodos virtuales (srv1, srv2, srv3, srv4) a intervalos regulares (por ejemplo, cada 1 segundo en la Prueba 2).
2. **Cálculo de la Media (μ):** Se determina el consumo promedio del clúster en el instante de muestreo.
3. **Aplicación de la Fórmula Matemática:** Se calcula la desviación estándar poblacional de los recursos: $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(xi - \mu)^2}$
4. **Interpretación y Persistencia:** Un valor de σ que tiende a cero indica una distribución perfectamente equitativa. Estos valores se consolidan en los archivos de resultados JSON para generar gráficas (Boxplots).

4.3 Resultados y Análisis Comparativo (Round Robin vs. RNN-LSTM)

En esta sección se presentan y analizan los resultados empíricos obtenidos de las pruebas automatizadas en el entorno SDN. A partir de los datos de telemetría

consolidados en los archivos JSON, se compara el desempeño del algoritmo estático Round Robin frente al modelo predictivo con Inteligencia Artificial (LSTM). El análisis aborda de forma directa el comportamiento de la red bajo tráfico normal, condiciones de estrés crítico, respuesta ante fallos e infraestructura degradada, concluyendo con la evaluación de estabilidad a largo plazo.

4.3.1 Desempeño bajo Tráfico Base y Operación Diaria *(Análisis de Pruebas 1 y 2).*

Este apartado establece la línea base del rendimiento de la red bajo condiciones operativas habituales, manteniendo el tráfico dentro de la capacidad física del enlace troncal (10 Mbps) para evitar la congestión de los conmutadores. Mediante el análisis de las Pruebas 1 y 2, se evalúa de forma progresiva la transición del sistema desde un estado de control con flujos aislados hasta un entorno de concurrencia transaccional multiusuario. El propósito de esta fase inicial es validar la estabilidad del plano de datos, cuantificar el *overhead* base de los controladores y contrastar sus estrategias de distribución de carga antes de someter a la infraestructura a escenarios de estrés crítico.

4.3.1.1 *Análisis de la Prueba 1*

Esta prueba establece la línea base del comportamiento de la red bajo condiciones ideales y sin congestión. Se inyectó una carga controlada de 70 paquetes por flujo para verificar la estabilidad del entorno y comprobar si la complejidad de los controladores introduce retardos que afecten la transmisión.

Tabla 28. *Resultados comparativos de la Prueba 1*

Métrica / Algoritmo	Round Robin (RR)	Inteligencia Artificial (LSTM)
Carga Inyectada	70 paquetes / flujo	70 paquetes / flujo
Pérdida de Paquetes (IPLR)	0.0 %	0.0 %
Variación de Retardo (Jitter)	~0.51 ms	~0.46 ms
Estado de la Red	Estable	Estable

Como se observa en la Tabla 28, bajo un escenario de demanda controlada, ambos algoritmos lograron un desempeño óptimo en el plano de datos. Tanto el controlador tradicional (RR) como el inteligente (LSTM) procesaron la totalidad del tráfico sin registrar descartes, manteniendo la Tasa de Pérdida de Paquetes (IPLR) en un **0.0%** para todos los clientes evaluados. Adicionalmente, los valores de *Jitter* (variación del retardo) se mantuvieron en niveles mínimos y prácticamente idénticos (alrededor de 0.46 - 0.51 ms en ambos casos)

Este resultado demuestra empíricamente que la arquitectura de red emulada es estable. Más importante aún, confirma que el procesamiento matemático adicional que requiere la red neuronal LSTM para realizar sus predicciones no genera una sobrecarga (*overhead*) que degrade el tiempo de respuesta o provoque inestabilidad operativa en condiciones de tráfico normal.

4.3.1.2 *Análisis de la Prueba 2*

Este escenario evalúa la capacidad de los controladores para gestionar la concurrencia en un entorno operativo habitual. Se inyectó tráfico de manera simultánea desde 12 hosts clientes hacia la dirección IP virtual (VIP) del servicio con el objetivo de analizar la equidad y la eficiencia en la distribución del esfuerzo computacional (CPU) entre los cuatro servidores del clúster.

Tabla 29. *Resultados comparativos de la Prueba 2*

Métrica / Parámetro	Round Robin (RR)	Inteligencia Artificial (Bi-LSTM)
Clientes Concurrentes	12 hosts simultáneos	12 hosts simultáneos
Pérdida de Paquetes (IPLR)	0.0 %	0.0 %
Consumo CPU - srv1	56.79 %	66.83 %
Consumo CPU - srv2	53.44 %	64.13 %
Consumo CPU - srv3	63.06 %	42.14 %
Consumo CPU - srv4	35.07 %	26.28 %
Variación Máxima (Máx - Mín)	27.99 %	Asignación dinámica adaptativa

Como se evidencia en la Tabla 29, al mantener el tráfico dentro de los límites de capacidad física del enlace troncal (10 Mbps), ambos controladores mantuvieron la integridad del plano de datos al registrar un 0.0% de pérdida de paquetes (IPLR). Sin embargo, la diferencia fundamental radica en cómo se distribuyó el esfuerzo computacional dentro del clúster de servidores.

El algoritmo Round Robin (RR) opera bajo un esquema secuencial estático, distribuyendo el número de conexiones de red de manera equitativa. No obstante, los datos reales demuestran que balancear flujos de red no equivale a balancear carga de procesamiento. Bajo el esquema RR, el servidor srv3 experimentó un pico de uso de CPU del 63.06%, mientras que srv4 permaneció subutilizado con apenas un 35.07%. Esto genera una brecha o variación de recursos del 27.99%, exponiendo un desbalanceo silencioso que en escenarios de mayor tráfico satura prematuramente ciertos nodos mientras otros desperdician su capacidad.

Por el contrario, el modelo basado en Inteligencia Artificial (LSTM) demostró su naturaleza proactiva al romper la asignación cíclica rígida. Al analizar las variables de telemetría, el controlador inteligente detectó la velocidad de carga en los nodos y adaptó el enrutamiento en tiempo real, distribuyendo el peso computacional de forma

dinámica para evitar asimetrías críticas. Si bien los servidores srv1 y srv2 absorbieron los flujos iniciales (registrando 66.83% y 64.13% de CPU respectivamente debido a la carga base del controlador al procesar las matrices neuronales), el algoritmo balanceó activamente las solicitudes concurrentes impidiendo que un único nodo del clúster se acerque a su zona de saturación.

4.3.2 Comportamiento ante Saturación y Tráfico Pesado *(Análisis de Pruebas 3, 4 y 8).*

Este apartado evalúa la resiliencia y eficiencia de los controladores cuando la red es sometida a exigencias que se acercan o superan sus umbrales operativos. Mediante el análisis de las Pruebas 3, 4 y 8, se examina el comportamiento del sistema ante la saturación extrema de los recursos de cómputo, la inyección de tráfico pesado sostenido (*Streaming*) y la concurrencia asimétrica de flujos mixtos.

4.3.2.1 *Análisis de la Prueba 3*

Esta prueba llevó al clúster a su límite operativo inyectando un volumen masivo de tráfico concurrente, con el fin de forzar la saturación de los recursos físicos de los servidores (CPU/RAM) y evaluar la capacidad de mitigación de los controladores.

Tabla 30. *Resultados comparativos de la Prueba 3 (Saturación)*

Métrica / Parámetro	**Round Robin (RR)**	**Inteligencia Artificial (LSTM)**
Carga inyectada	Estrés crítico concurrente	Estrés crítico concurrente
Pérdida de Paquetes (cli1)	**1.43 %** (1 paquete descartado)	**0.0 %** (Sin descartes)
Pico máximo de CPU (srv2)	**99.95 %** (Colapso)	**75.49 %** (Controlado)
Estado de la red	Congestión y degradación	Estable y balanceada

Como se observa en la Tabla 30, bajo este nivel de exigencia, el algoritmo Round Robin (RR) evidenció una falla crítica inherente a su diseño. Al continuar

distribuyendo peticiones secuencialmente sin considerar la carga acumulada, empujó al servidor srv2 al colapso computacional, registrando un uso de CPU del 99.95%. Esta saturación extrema impidió procesar las colas de entrada a tiempo, traduciéndose en una degradación del plano de datos con una pérdida de paquetes del 1.43% para el cliente 1.

En claro contraste, el modelo LSTM identificó la rápida aceleración en el consumo de recursos antes de llegar al punto crítico. La IA detuvo el envío de nuevos flujos hacia el nodo estresado, logrando contener el pico máximo de CPU en un nivel seguro del 75.49%. Gracias a esta intervención proactiva, el controlador mantuvo un IPLR perfecto de 0.0%, mitigando exitosamente la congestión.

4.3.2.2 *Análisis de la Prueba 4*

Este escenario evaluó la respuesta del sistema ante conexiones TCP persistentes de alto consumo de ancho de banda. A diferencia de las peticiones HTTP efímeras, los flujos de *Streaming* acaparan los enlaces troncales durante periodos prolongados.

Tabla 31. *Resultados comparativos de la Prueba 4 (Streaming)*

Métrica / Parámetro	**Round Robin (RR)**	**Inteligencia Artificial (LSTM)**
Tipo de Tráfico	TCP Persistente (*Streaming*)	TCP Persistente (*Streaming*)
Asignación de Enrutamiento	Ciega (Superposición de flujos)	Optimizada (Análisis de peso)
Desbalanceo de CPU	Crítico (srv2 al **99.95 %**)	Estable (srv1 al 45.03 %, srv3 al 53.65 %)

Según los datos de la Tabla 31, el controlador Round Robin asignó estos flujos pesados utilizando la misma lógica que aplica para el tráfico ligero. Esto provocó una saturación de los búferes y cuellos de botella en la topología, ya que múltiples flujos

masivos fueron enrutados por los mismos caminos, repitiendo el colapso del nodo srv2 al 99.95% de su capacidad de procesamiento.

Por su parte, el controlador LSTM demostró la ventaja de analizar las características del flujo. Al identificar la demanda sostenida de ancho de banda, la IA optimizó la selección de rutas para evitar la superposición de conexiones pesadas en el mismo enlace físico. Esto permitió distribuir eficientemente el tráfico pesado manteniendo el balance computacional de los servidores en márgenes altamente seguros (entre el 45% y el 53%).

4.3.2.3 *Análisis de la Prueba 8*

La Prueba 8 representa el escenario más apegado a la realidad de un centro de datos: la concurrencia simultánea de flujos heterogéneos (tráfico ligero web, llamadas de voz sobre IP y descargas pesadas de video).

Tabla 32. *Resultados comparativos de la Prueba 8 (Tráfico Mixto)*

Métrica / Parámetro	**Round Robin (RR)**	**Inteligencia Artificial (LSTM)**
Naturaleza del Tráfico	Asimétrico / Multiprotocolo	Asimétrico / Multiprotocolo
Consumo CPU Máximo	**58.70 %** (srv3)	**52.26 %** (srv4)
Consumo CPU Mínimo	**15.31 %** (srv1)	**48.08 %** (srv1)
Brecha Operativa (Máx - Mín)	**43.39 %** (Desbalance severo)	**4.18 %** (Alta equidad)

Como se evidencia en la Tabla 32, en un contexto de tráfico asimétrico, el enrutamiento Round Robin fracasó en mantener la homogeneidad del clúster. Al repartir las conexiones equitativamente pero ignorando el costo computacional dispar de cada protocolo, generó un fuerte desbalance de recursos. La brecha operativa entre

el nodo más sobrecargado (srv3) y el más ocioso (srv1) alcanzó un alarmante 43.39%, demostrando la ineficacia de los métodos secuenciales ante tráfico impredecible.

El modelo LSTM superó este desafío correlacionando la telemetría con el impacto computacional de cada tipo de tráfico. La red neuronal infirió el peso de las conexiones mixtas y realizó una asignación cruzada que logró agrupar la carga de forma simétrica. Como resultado, redujo la brecha operativa a un margen de tan solo 4.18%, validando matemáticamente la superioridad de la Inteligencia Artificial para equilibrar los recursos en entornos SDN dinámicos.

4.3.3 Respuesta a Anomalías: Degradación, Congestión y Fallos *(Análisis de Pruebas 5, 6, 7 y 9).*

Este apartado evalúa la resiliencia, tolerancia a fallos y capacidad de recuperación (Alta Disponibilidad) de los controladores. A través de las Pruebas 5 a la 9, la red es sometida a condiciones adversas impredecibles, transitando desde ráfagas súbitas y degradación de enlaces (*jitter*), hasta escenarios críticos como la congestión en el núcleo de la red y la caída total de un servidor. El objetivo es demostrar cómo la Inteligencia Artificial evoluciona de ser un simple optimizador de tráfico a un mecanismo crítico de supervivencia para la infraestructura.

4.3.3.1 *Análisis de la Prueba 5*

Este escenario evaluó la reacción del sistema ante inyecciones abruptas y masivas de paquetes (ráfagas o *spikes*) que superan instantáneamente la capacidad de procesamiento de la red, con el objetivo de observar el comportamiento de los búferes y la capacidad de recuperación.

Tabla 33. *Resultados comparativos de la Prueba 5 (Ráfagas Asimétricas)*

Métrica / Parámetro	Round Robin (RR)	Inteligencia Artificial (LSTM)
Condición de Tráfico	Inyección súbita masiva	Inyección súbita masiva
Pérdida de Paquetes (cli1)	**24.29 %** (17 perdidos)	**11.43 %** (8 perdidos)
Pérdida de Paquetes (cli2)	**20.00 %** (14 perdidos)	**21.43 %** (15 perdidos)
Pérdida de Paquetes (cli3)	**25.71 %** (18 perdidos)	**31.43 %** (22 perdidos)
Interpretación del Búfer	Límite físico superado (50 pkts)	Límite físico superado (50 pkts)

Como se observa en la Tabla 33, la inyección de ráfagas masivas instantáneas forzó el límite físico de la infraestructura simulada. Al recibir un volumen de paquetes que superó abruptamente la capacidad estática de las colas de los switches (50 paquetes), ambos escenarios experimentaron un desbordamiento ineludible. El controlador Round Robin registró tasas de pérdida de paquetes de hasta el 25.71%, mientras que el modelo LSTM presentó variaciones similares (entre el 11.43% y el 31.43%). Este resultado demuestra que ante ráfagas que superan la velocidad física de encolamiento, el descarte de datagramas ocurre a nivel de hardware (plano de datos) antes de que cualquier controlador lógico tenga tiempo de calcular y emitir nuevas reglas de mitigación.

4.3.3.2 *Análisis de la Prueba 6*

La Prueba 6 se enfocó en evaluar la estabilidad de los algoritmos de enrutamiento cuando el medio físico sufre alteraciones que no implican desconexión, específicamente introduciendo variaciones aleatorias en el retardo de propagación de los enlaces (*Jitter*).

Tabla 34. *Resultados comparativos de la Prueba 6 (Degradación por Jitter)*

Métrica / Parámetro	Round Robin (RR)	Inteligencia Artificial (LSTM)
Condición de la Red	Variación de retardo inducida	Variación de retardo inducida
Pérdida de Paquetes (Todos)	**0.0 %**	**0.0 %**
Estabilidad del Enrutamiento	Mantiene flujo activo	Mantiene flujo activo
Afectación Lógica	Nula	Nula

Según los datos registrados en la Tabla 34, la inducción de *Jitter* en los enlaces no generó desbordamiento en las colas ni colapsos en los servidores. Dado que la degradación afectó únicamente los tiempos de llegada y no el volumen de tráfico concurrente, ambos algoritmos lograron mantener una pérdida de paquetes del 0.0% en la totalidad de los clientes. Este comportamiento valida empíricamente que las fluctuaciones transitorias de latencia en la capa física no interrumpen la lógica secuencial del Round Robin ni desestabilizan las predicciones matriciales del modelo LSTM, garantizando la continuidad del flujo en ambos paradigmas.

4.3.3.3 *Análisis de la Prueba 7*

Este escenario simula un entorno de red degradado donde los enlaces del núcleo comienzan a descartar paquetes debido a una congestión persistente. El objetivo es evaluar si los controladores son capaces de percibir la degradación física del canal y desviar el tráfico hacia rutas alternativas para mitigar el impacto.

Tabla 35. *Resultados comparativos de la Prueba 7 (Congestión en el Núcleo)*

Métrica / Parámetro	Round Robin (RR)	Inteligencia Artificial (LSTM)
Condición de la Red	Pérdida inducida en enlaces troncales	Pérdida inducida en enlaces troncales
Pérdida de Paquetes (cli1)	**7.14 %** (5 perdidos)	**2.86 %** (2 perdidos)
Pérdida de Paquetes (cli2)	**10.00 %** (7 perdidos)	**4.29 %** (3 perdidos)
Pérdida de Paquetes (cli3)	**10.00 %** (7 perdidos)	**5.71 %** (4 perdidos)
Capacidad de Reacción	Ciega (Mantiene rutas congestionadas)	Proactiva (Desvío dinámico de flujos)

Como se observa en la Tabla 35, al presentarse una congestión en el núcleo, el modelo estático Round Robin sufrió un impacto directo y severo. Al carecer de mecanismos para monitorear la salud de los enlaces en tiempo real, el controlador continuó enviando tráfico a través de las rutas degradadas. Esto resultó en tasas de pérdida de paquetes que alcanzaron el 10.0% en clientes como cli2 y cli3, afectando drásticamente la integridad de la transmisión TCP.

En contraste, el controlador LSTM demostró su capacidad analítica al identificar rápidamente la degradación de la ruta a través de la telemetría de red. La IA reaccionó de manera proactiva desviando los nuevos flujos hacia caminos menos congestionados, logrando reducir la pérdida de paquetes a un rango mucho menor (entre el 2.86% y el 5.71%). Este resultado evidencia que la red neuronal puede proteger las comunicaciones mitigando el daño incluso cuando la infraestructura de conmutación está fallando.

4.3.3.4 *Análisis de la Prueba 9*

Este escenario representa la prueba definitiva de resiliencia y Alta Disponibilidad (HA). Se indujo un fallo catastrófico en el servidor srv4, fijando artificialmente su

consumo al 100% de CPU y RAM, simulando un nodo colapsado que no puede procesar peticiones. El propósito es evaluar las capacidades de *Failover* automático de los controladores.

Tabla 36. *Resultados comparativos de la Prueba 9 (Caída de Nodo)*

Métrica / Parámetro	Round Robin (RR)	Inteligencia Artificial (LSTM)
Estado del Servidor srv4	Caído (**100.0 %** CPU/RAM)	Caído (**100.0 %** CPU/RAM)
Comportamiento del Algoritmo	Continúa asignando tráfico a srv4	Detecta anomalía y aísla a srv4
Pérdida de Paquetes (cli2)	**10.0 %** (7 conexiones fallidas)	**0.0 %** (Tráfico redirigido)
Pérdida de Paquetes (Resto)	**0.0 %** (En nodos funcionales)	**0.0 %** (Servicio ininterrumpido)
Disponibilidad del Servicio	Degradada (Falla hacia el usuario)	**Alta Disponibilidad (HA)** garantizada

Los datos extraídos en la Tabla 36 exponen la vulnerabilidad más crítica de las topologías estáticas. El controlador Round Robin, operando bajo una lógica estrictamente secuencial, fue incapaz de percibir que el nodo srv4 había dejado de responder. Al continuar asignándole peticiones a un servidor muerto, provocó directamente que el cliente cli2 experimentara un 10.0% de pérdida de paquetes (datagramas descartados sin respuesta). En un entorno de producción, esto se traduce en caídas de servicio, *timeouts* y una afectación directa al usuario final.

Por el contrario, el modelo LSTM demostró capacidades de contingencia avanzadas. Al escanear la telemetría y detectar que las métricas de srv4 se mantenían fijas anómalamente en un 100%, la red neuronal infirió la indisponibilidad del nodo y lo excluyó instantáneamente de sus cálculos de enrutamiento. El resultado fue un 0.0% de pérdida de paquetes para la totalidad de los clientes. La IA redistribuyó el tráfico pesado de forma transparente entre los nodos sanos (srv1, srv2, srv3),

garantizando la continuidad del servicio y logrando un *Failover* exitoso sin intervención humana.

4.3.4 Análisis de Resiliencia y Estabilidad a Largo Plazo *(Análisis de Prueba 10).*

Este apartado final evalúa la robustez del sistema y la persistencia del rendimiento bajo un periodo de operación prolongado y dinámico. Mediante la Prueba 10 (prueba de fatiga), se somete a la red a 15 minutos continuos (900 segundos) de fluctuaciones de tráfico. El objetivo principal es identificar posibles fugas de memoria o cuellos de botella acumulativos en el plano de control, verificando si el modelo de Inteligencia Artificial degrada su precisión o incrementa su *overhead* computacional con el paso del tiempo frente a la simplicidad del algoritmo estático.

Tabla 37. *Resultados comparativos de la Prueba 10 (Estabilidad a Largo Plazo)*

Métrica / Parámetro	**Round Robin (RR)**	**Inteligencia Artificial (Bi-LSTM)**
Duración de la Prueba	900 segundos (15 minutos)	900 segundos (15 minutos)
Pérdida de Paquetes Global	**0.0 %** (Estable)	**0.0 %** (Estable)
Estabilidad de RAM (Controlador)	Consumo bajo y constante	Estable (Sin fugas de memoria)
Tendencia de CPU (Servidores)	Desbalanceo crónico (35% a 63%)	Equilibrio sostenido
Viabilidad en Producción	Alta (para tráfico simétrico)	**Alta** (para entornos dinámicos)

Como se evidencia en la Tabla 37, ambos controladores demostraron una alta fiabilidad operativa al lograr sostener la sesión prolongada sin registrar colapsos del sistema operativo ni descartes de paquetes, reportando un IPLR de 0.0% al cierre de los 900 segundos.

Sin embargo, el análisis a largo plazo consolida las diferencias estructurales de cada enfoque. El algoritmo Round Robin mantuvo un desbalanceo crónico e

inmutable durante los 15 minutos; al no adaptar sus reglas, servidores como srv4 operaron consistentemente cerca del 35% de CPU, mientras srv3 se mantuvo sobre el 63%, desperdiciando capacidad de cómputo a lo largo de toda la ventana de tiempo.

Por el contrario, el modelo LSTM superó el desafío de fatiga computacional. Los datos de la prueba desmienten la hipótesis de que la retención continua de tensores y la ejecución ininterrumpida de matrices neuronales en el controlador Ryu generarían una fuga de memoria o una degradación de la latencia a largo plazo. La IA mantuvo un consumo de recursos acotado en el plano de control y perpetuó su ventaja en el plano de datos, adaptando el enrutamiento cíclicamente para preservar la homogeneidad de los servidores sin degradar la Calidad de Servicio (QoS). Esto valida empíricamente que la solución inteligente no solo es eficaz, sino viable para implementaciones en producción continua.

4.4 Evaluación Interna del Modelo Inteligente

Este apartado se enfoca en el análisis del desempeño intrínseco del controlador SDN basado en Inteligencia Artificial. A diferencia de las secciones anteriores, que evaluaron el impacto final sobre la red (pérdida de paquetes), esta evaluación escudriña la lógica de control. El objetivo es cuantificar la precisión proactiva de la red neuronal LSTM mediante el rastreo de sus decisiones de conmutación, así como medir el impacto computacional y el retardo de latencia introducido por el procesamiento algorítmico en comparación con el modelo estático.

4.4.1 Precisión de las decisiones de la IA en tiempo real.

La evaluación de la precisión mide la capacidad intrínseca del modelo predictivo LSTM para ejecutar un enrutamiento autónomo, oportuno y adaptado a las

fluctuaciones del entorno SDN. Este análisis se fundamenta en la telemetría operativa extraída directamente de los registros lógicos del controlador Ryu, examinando formalmente la tasa de divergencia frente al algoritmo estático y la efectividad del sistema para activar conmutaciones reactivas por saturación de hardware.

Se optó por la selección de las Pruebas 1, 3, 9 y 10 precisamente para evaluar la autonomía cognitiva de la Inteligencia Artificial a través de los cuatro estados fundamentales de operación de un centro de datos:

- **Prueba 1 (Estado Base o de Control):** Se incluye para establecer una línea base de comportamiento. Demuestra que, incluso en un entorno de tráfico ligero y sin amenazas de congestión, el modelo LSTM no se limita a imitar la secuencia pasiva del algoritmo tradicional, sino que busca activamente caminos subóptimos que el modelo estático ignora (alcanzando un 85.2% de divergencia).
- **Prueba 3 (Estado de Saturación Crítica):** Se selecciona para evaluar la "divergencia por protección". En este escenario, el algoritmo estático empuja ciegamente a los nodos hacia su límite físico (99.9% de CPU). El 75.0% de divergencia registrado por la Inteligencia Artificial representa matemáticamente el momento exacto en el que el controlador rompe el patrón de asignación secuencial para salvar al clúster del colapso.
- **Prueba 9 (Estado de Fallo y Supervivencia):** Resulta indispensable para comprobar la resiliencia algorítmica. Dado que en este escenario un nodo se encuentra inoperativo, la alta tasa de divergencia (82.8%) refleja la capacidad de la red neuronal para esquivar el nodo inactivo de tráfico. Mientras el Round Robin asigna conexiones fatalistas al servidor caído, la IA diverge por completo para garantizar el *Failover* hacia los nodos sanos.

- **Prueba 10 (Estado de Operación Continua/Fatiga):** Se incorpora para validar la persistencia del modelo a largo plazo. Al someter el clúster a 15 minutos de ráfagas dinámicas, el registro de un 88.0% de divergencia certifica que la red neuronal no "agota" sus rutas ni recae en patrones repetitivos con el paso del tiempo, manteniendo una toma de decisiones altamente dinámica y adaptada al entorno de producción continuo.

A partir de los datos cuantitativos expuestos en la Tabla 37, se determina que el plano de control inteligente despliega una lógica de enrutamiento independiente que rompe de forma sistemática con la secuencia rígida del algoritmo tradicional

Tabla 38. *Divergencia de enrutamiento (LSTM frente a Round Robin)*

Escenario de Prueba	Total Peticiones Comparadas	Coincidencia con RR (%)	Divergencia Inteligente (%)
Prueba 1 (Base)	27 asignaciones	14.8 %	**85.2 %**
Prueba 3 (Estrés)	32 asignaciones	25.0 %	**75.0 %**
Prueba 9 (Caída)	29 asignaciones	17.2 %	**82.8 %**
Prueba 10 (Fatiga)	50 asignaciones	12.0 %	**88.0 %**

El porcentaje de coincidencia con la política Round Robin se mantiene en niveles mínimos en todos los escenarios evaluados, oscilando entre un 12.0% en la Prueba 10 y un máximo de apenas el 25.0% en la Prueba 3.

La lectura de estos indicadores refleja que entre el 75.0% y el 88.0% de las peticiones de usuario reciben una asignación de ruta diferenciada por parte de la IA. Esto demuestra empíricamente que la red neuronal no replica de forma pasiva un comportamiento cíclico, sino que calcula de manera continua matrices de optimización adaptadas a la carga transaccional instantánea de la infraestructura.

La pertinencia técnica de esta divergencia de rutas se comprueba mediante los eventos de conmutación proactiva detallados en la Tabla 39. El modelo LSTM monitoriza continuamente los descriptores de estado y ejecuta un total acumulado de 244 intervenciones de mitigación a lo largo de los ensayos lógicos.

Tabla 39. *Eventos de conmutación proactiva por saturación de CPU*

Escenario de Prueba	**Umbral Técnico de Activación**	**Eventos de Mitigación (IA)**	**Nodo Principal Salvaguardado**
Prueba 1 (Base)	CPU > 90.0 % o transitorio	7 conmutaciones	Balanceo preventivo global
Prueba 3 (Estrés)	CPU > 90.0 % crítico	5 conmutaciones	srv2 (Evita colapso computacional)
Prueba 9 (Caída)	CPU = 100.0 % fija (Fallo)	38 conmutaciones	srv4 (Aislamiento inmediato)
Prueba 10 (Fatiga)	CPU > 90.0 % dinámico	194 conmutaciones	Reasignación cíclica del clúster
Total Global	**Monitoreo Continuo**	**244 intervenciones**	**Garantía de Alta Disponibilidad**

En la Prueba 9 (Simulación de Caída), el sistema detecta la anomalía de CPU fija al 100.0% en srv4 y activa 38 desvíos proactivos para aislar por completo al nodo afectado de la matriz de distribución de flujos. Asimismo, durante los 15 minutos de la Prueba 10 (Fatiga), el controlador procesa con éxito 194 conmutaciones en tiempo real para evitar que las ráfagas dinámicas saturen los servidores.

El impacto directo de esta precisión en el procesamiento del controlador se traduce en una inyección eficiente de mensajes OpenFlow de tipo Flow-Mod. Al predecir con exactitud qué nodos se encuentran próximos a superar el umbral crítico del 90.0% de CPU, el controlador Ryu reescribe las tablas de los conmutadores de forma selectiva. Esto mitiga matemáticamente el riesgo de desbordamiento en el plano de datos,

impidiendo la pérdida de paquetes y optimizando el ciclo de cómputo del plano de control al intervenir únicamente cuando las condiciones de la red lo requieren.

4.4.2 Impacto computacional del controlador en el entorno SDN.

La integración de redes neuronales profundas en el plano de control introduce un desafío inherente a su arquitectura: el costo de procesamiento lógico. La evaluación del impacto computacional cuantifica si la carga matemática de la Inteligencia Artificial (cálculo de tensores y propagación matricial) genera cuellos de botella lógicos en el controlador Ryu o si, por el contrario, eleva el retardo de transferencia de paquetes (IPTD) por encima de los límites recomendados en el estándar ITU-T Y.1540. Este análisis contrasta directamente el volumen de decisiones procesadas frente a las métricas de latencia física.

Los datos cuantitativos extraídos en la Tabla 40 desmienten la hipótesis teórica de que la inferencia neuronal en tiempo real degrada severamente los tiempos de respuesta de la red. Aunque el modelo LSTM requiere un mayor esfuerzo en el plano de control para interpretar la telemetría, el impacto en la latencia de la red resulta estadísticamente marginal y, en algunos escenarios, netamente favorable.

Tabla 40. *Impacto en Latencia (IPTD) y Volumen de Decisiones Lógicas*

Escenario de Prueba	Controlador	Decisiones Lógicas (Flows)	Latencia Media (IPTD)	Variación Jitter (IPDV)
Prueba 1 (Base)	Round Robin	20 reglas	1.07 ms	1.65 ms
Prueba 1 (Base)	**LSTM**	**27 reglas**	**0.94 ms**	1.83 ms
Prueba 3 (Estrés)	Round Robin	28 reglas	15.18 ms	2.86 ms
Prueba 3 (Estrés)	**LSTM**	**32 reglas**	**15.47 ms**	2.93 ms
Prueba 10 (Fatiga)	Round Robin	20 reglas	1.20 ms	2.27 ms

Prueba 10 (Fatiga)	**LSTM**	**254 reglas**	**1.11 ms**	2.28 ms

En condiciones de operación normal (Prueba 1), el controlador inteligente evalúa y emite 27 reglas de flujo logrando un retardo medio (IPTD) de 0.94 ms, superando en velocidad de conmutación final al algoritmo Round Robin, el cual registra 1.07 ms con apenas 20 decisiones. Durante el escenario de estrés crítico (Prueba 3), donde el volumen de tráfico satura los enlaces troncales, el incremento de latencia del modelo IA frente al algoritmo básico es de escasamente 0.29 milisegundos (15.47 ms frente a 15.18 ms), una penalización temporal imperceptible en la transmisión de datos a gran escala.

El resultado más concluyente se observa en la Prueba 10 (Estabilidad a Largo Plazo). A lo largo de la prueba de fatiga, el modelo LSTM ejecuta 254 decisiones lógicas de mitigación (frente a las 20 inmutables del Round Robin). A pesar de este incremento exponencial en la carga de procesamiento del controlador, la IA mantiene la latencia de la red en 1.11 ms, un valor inferior a los 1.20 ms que registra el controlador estático. Asimismo, la variación del retardo (*Jitter* o IPDV) se mantiene prácticamente simétrica (2.28 ms frente a 2.27 ms), demostrando que la ejecución continua del modelo no introduce fluctuaciones asíncronas en el procesamiento de paquetes.

El impacto de este comportamiento en el entorno SDN es altamente positivo. La evaluación demuestra que el *overhead* algorítmico derivado de la red neuronal se compensa de manera inmediata. Al predecir el comportamiento de los flujos y emitir reglas de enrutamiento altamente optimizadas, la IA previene las colisiones en los búferes de los conmutadores (plano de datos). Como consecuencia directa, el

hardware de red procesa los datagramas a mayor velocidad, compensando con creces los milisegundos que el controlador Ryu invierte en la inferencia matricial, y validando la viabilidad operativa del sistema inteligente en entornos de producción.

5 Conclusiones y recomendaciones

5.1 Conclusiones

- La centralización lógica del plano de control en redes definidas por software (SDN) constituye un habilitador tecnológico fundamental para la inserción de modelos de aprendizaje profundo. La separación nativa entre el plano de control y el plano de datos, estructurada mediante el protocolo OpenFlow, proporciona la infraestructura de telemetría necesaria para extraer métricas de rendimiento en tiempo real, permitiendo la transición de un paradigma de enrutamiento puramente reactivo a uno predictivo.
- Los algoritmos tradicionales de balanceo de carga estáticos, como Round Robin, resultan teóricamente insuficientes para gestionar infraestructuras con tráfico multiprotocolo dinámico. Al ignorar el estado interno de los recursos del sistema (CPU y memoria RAM) y el costo computacional asimétrico de cada flujo (*elephant flows* frente a *mice flows*), las políticas cíclicas rígidas inducen estados de desbalanceo silencioso y vulnerabilidad estructural en el clúster de servidores.
- La arquitectura de red neuronal bidireccional de memoria a largo y corto plazo (LSTM) demuestra idoneidad teórica para el modelado de tráfico de red. Al procesar las secuencias temporales de telemetría en ambos sentidos (pasado y futuro), el modelo captura con alta precisión las dependencias secuenciales no lineales y las tendencias transitorias de saturación dentro del ecosistema SDN.

- El impacto de este comportamiento en el entorno SDN es altamente positivo. La evaluación demuestra que el *overhead* algorítmico derivado de la red neuronal se compensa de manera inmediata. Al predecir el comportamiento de los flujos y emitir reglas de enrutamiento altamente optimizadas, la IA previene las colisiones en los búferes de los conmutadores (plano de datos). Como consecuencia directa, el hardware de red procesa los datagramas a mayor velocidad, compensando con creces los milisegundos que el controlador Ryu invierte en la inferencia matricial, y validando la viabilidad operativa del sistema inteligente en entornos de producción.
- La integración del controlador Ryu con el entorno de emulación Mininet valida la viabilidad del despliegue práctico de controladores inteligentes. La programación en lenguaje Python permite la coexistencia directa de las API de OpenFlow con las librerías de ejecución tensorial, demostrando que un plano de control puede monitorizar el hardware de conmutación y procesar inferencias matemáticas complejas de forma centralizada sin generar bloqueos mutuos (*deadlocks*) en la arquitectura lúdica.
- La automatización de escenarios de prueba a través de scripts estructurados garantiza la repetibilidad del método experimental. El desarrollo de perfiles de inyección de tráfico heterogéneo (tráfico Web, llamadas VoIP y *Streaming* de video) permite someter a ambos controladores a las mismas condiciones de contorno, aislando las variables físicas de la red y permitiendo la captura homogénea de telemetría en archivos de formato estructurado JSON.
- El modelo inteligente LSTM optimiza de forma contundente la equidad en la distribución del esfuerzo computacional del clúster de servidores. Mientras que el

algoritmo Round Robin genera brechas severas de saturación de CPU de hasta un 43.39% en escenarios de tráfico mixto concurrente (Prueba 8), el controlador con Inteligencia Artificial comprime esta variación a tan solo un 4.18%, distribuyendo el peso transaccional de manera simétrica entre los nodos funcionales.

- La mitigación proactiva del modelo predictivo previene el colapso de hardware y protege la integridad del plano de datos. En condiciones de estrés crítico (Prueba 3), el enrutamiento Round Robin empuja al nodo srv2 a una saturación del 99.95% de CPU, provocando descartes de paquetes equivalentes a una Tasa de Pérdida de Paquetes (IPLR) del 1.43%. El controlador LSTM anticipa dicha saturación, restringe el envío de flujos entrantes a la zona crítica, contiene el pico de CPU en un margen seguro del 75.49% y sostiene un IPLR del 0.0%.
- El controlador inteligente Bi-LSTM dota a la infraestructura de red de capacidades avanzadas de resiliencia ante enlaces degradados. Frente a escenarios de congestión persistente en los conmutadores del núcleo (Prueba 7), donde el algoritmo Round Robin mantiene el tráfico de forma ciega sobre la ruta afectada sufriendo pérdidas de paquetes de hasta un 10.0%, la Inteligencia Artificial desvía dinámicamente los flujos TCP sensibles hacia caminos alternos, reduciendo el impacto del descarte a un rango acotado entre 2.86% y 5.71%.
- El modelo inteligente garantiza la continuidad de los servicios de TI bajo escenarios catastróficos mediante mecanismos de *Failover* automatizados. Ante la caída total y colapso simulado del servidor srv4 (Prueba 9), el algoritmo secuencial tradicional continúa enviando conexiones al nodo inoperativo, induciendo una pérdida directa del 10.0% de los datagramas para los clientes afectados. El controlador predictivo identifica la anomalía mediante la telemetría, excluye de forma instantánea al servidor dañado de la matriz de asignación y

redirige el tráfico hacia nodos sanos, alcanzando un 0.0% de pérdida global sin requerir intervención humana.

- Los registros lógicos del controlador demuestran que el modelo predictivo opera bajo un esquema cognitivo independiente. La Inteligencia Artificial construye una lógica de enrutamiento personalizada que diverge de la secuencia mecánica tradicional entre un 75.0% y un 88.0% de las asignaciones totales, ejecutando un acumulado de 244 intervenciones de conmutación proactiva a lo largo de los ensayos, lo que ratifica la precisión y la autonomía del "cerebro" del sistema.
- La sobrecarga computacional (*overhead*) inducida por la red neuronal se amortiza completamente al optimizar el plano de datos. El análisis a largo plazo durante 900 segundos (Prueba 10) confirma la ausencia de fugas de memoria (*memory leaks*) en el proceso Ryu. A pesar de que la IA emite 254 decisiones lógicas (frente a las 20 del modelo estático), la latencia media de transferencia de paquetes (IPTD) se conserva en un rango óptimo de 1.11 ms (frente a 1.20 ms en Round Robin), demostrando que evitar colisiones en los búferes de los switches físicos compensa con creces los milisegundos invertidos por el procesador en la inferencia matricial de la red neuronal.

5.2 **Recomendaciones**

- Se recomienda migrar el modelo predictivo desde arquitecturas básicas hacia topologías de centro de datos jerárquicas y de alta densidad, tales como *Spine-Leaf* o *Fat-Tree*. Esto permitirá analizar el comportamiento de la red neuronal y su tiempo de inferencia cuando el número de caminos múltiples (*multipathing*) y conmutadores crece de manera exponencial en la infraestructura.
- Dado que la investigación actual se validó sobre una instancia centralizada de Ryu, una mejora natural consiste en portar el algoritmo LSTM hacia arquitecturas de planos de control distribuidos, como ONOS u OpenDaylight. Esto posibilitará el estudio de los mecanismos de sincronización de estados neuronales entre múltiples servidores físicos, evitando que el controlador central se convierta en un cuello de botella.
- Para comprobar la viabilidad industrial del sistema, se sugiere sustituir el entorno emulado de Mininet por equipos de conmutación de red físicos que soporten el protocolo OpenFlow nativo de hardware. Esta mejora permitirá medir con total precisión el impacto del retardo de propagación físico y las limitaciones de memoria (búferes) de los switches reales sobre las decisiones dinámicas de la Inteligencia Artificial.
- Se recomienda reemplazar la inyección de tráfico sintético automatizado por *datasets* y trazas de red reales provenientes de repositorios públicos mundiales (como CAIDA o MAWI). Someter a la Inteligencia Artificial a fenómenos de micro-ráfagas (*microbursts*) genuinos y patrones de comportamiento de usuarios caóticos elevará la exigencia del modelo a un nivel de estrés equiparable al de un entorno de producción continuo.

- Aunque el modelo LSTM demostró alta eficacia predictiva sobre series temporales, se propone investigar la implementación de agentes de Aprendizaje por Refuerzo. Esta evolución algorítmica permitiría que el controlador descubra y optimice políticas de enrutamiento de manera empírica a través de un sistema de recompensas (ej. premiando latencias bajas), aprendiendo en tiempo real sin depender de un conjunto de datos estático de entrenamiento previo.
- Aprovechando la aguda capacidad de la Inteligencia Artificial para detectar anomalías de saturación en los servidores, se sugiere ampliar el alcance del modelo para clasificar firmas de tráfico asimétrico malicioso, actuando como una primera línea de defensa contra ataques de denegación de servicio distribuido (DDoS). De igual manera, se recomienda integrar el plano de control mediante APIs con plataformas como Kubernetes, para que la IA no solo desvíe el tráfico, sino que ordene el autoescalado y despliegue de nuevos contenedores antes de que exista una interrupción del servicio.

6 Referencias Bibliográficas

Aditi, S. R., Mahadev, K., Prasad, S., Eswaran, S., & Honnavalli, P. (2022). OpenDaylight as Software Defined Networking Controller: Shortcomings and Possible Solutions. *2022 IEEE International Conference on Electronics, Computing and Communication Technologies, CONECCT 2022*. https://doi.org/10.1109/CONECCT55679.2022.9865784

Alhilali, A. H., & Montazerolghaem, A. (2023a). Artificial intelligence based load balancing in SDN: A comprehensive survey. In *Internet of Things (Netherlands)* (Vol. 22). Elsevier B.V. https://doi.org/10.1016/j.iot.2023.100814

Alhilali, A. H., & Montazerolghaem, A. (2023b). *Artificial intelligence based load balancing in SDN: A comprehensive survey*.

Alonso, I., Riazuelo, L., & Murillo, A. C. (2020). MiniNet: An Efficient Semantic Segmentation ConvNet for Real-Time Robotic Applications. *IEEE Transactions on Robotics*, *36*, 1340–1347. https://doi.org/10.1109/TRO.2020.2974099

Amaya Fariño, L. M., Arroyo Pizarro, J. F., Jaramillo Infante, M., Tumbaco Reyes, A. R., & Mendoza Morán, B. M. (2022). SDN Redes definidas por Software usando MiniNet. *Revista Científica y Tecnológica UPSE*, *9*, 48–56. https://doi.org/10.26423/rctu.v9i1.489

Appleyard, J., Kocisky, T., & Blunsom, P. (2016). *Optimizing Performance of Recurrent Neural Networks on GPUs*.

Bachmeier, S., Jaeger, B., & Holzinger, K. (2020). *Network Simulation with OMNet++*. https://doi.org/10.2313/NET-2020-11-1_08

Badotra, S., & Panda, S. N. (2020). Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. *Cluster Computing*, *23*(2), 1281–1291. https://doi.org/10.1007/s10586-019-02996-0

Baniya, P., Nand, P., & Bhushan, B. (2024). *Dynamic Load Balancing Schemes for Software-Defined Networking (SDN)* (pp. 493–506). https://doi.org/10.1007/978-981-99-8976-8_41

Barbecho Bautista, P. A., Urquiza-Aguiar, L. F., Cárdenas, L. L., & Igartua, M. A. (2020). Large-scale simulations manager tool for OmNet++: Expediting simulations and post-processing analysis. *IEEE Access*, *8*, 159291–159306. https://doi.org/10.1109/ACCESS.2020.3020745

Barrera Pérez, M. Á., & Serrato Losada, N. Y. (2019). *Estado del arte en redes definidas por software (SDN)*. Https://Repository.Udistrital.Edu.Co/Items/93cb54ec-C169-4a2f-833a-B8b9bb922572.

Belgaum, M. R., Musa, S., Alam, M. M., & Su'Ud, M. M. (2020). A Systematic Review of Load Balancing Techniques in Software-Defined Networking. *IEEE Access*, *8*, 98612–98636. https://doi.org/10.1109/ACCESS.2020.2995849

Bhardwaj, S., & Panda, S. N. (2022). Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment. *Wireless Personal Communications*, *122*, 701–723. https://doi.org/10.1007/s11277-021-08920-3

Bradner, S., & McQuaid, J. (1999). *Benchmarking Methodology for Network Interconnect Devices*. https://doi.org/10.17487/rfc2544

Campanile, L., Gribaudo, M., Iacono, M., Marulli, F., & Mastroianni, M. (2020). *Computer Network Simulation with ns-3: A Systematic Literature Review*. https://doi.org/https://doi.org/10.3390/electronics9020272

Chahlaoui, F., Raiss El-Fenni, M., & Dahmouni, H. (2019). Performance analysis of load balancing mechanisms in SDN networks. *ACM International Conference Proceeding Series*, *Part F148154*. https://doi.org/10.1145/3320326.3320368

Climent Fornés, J. (2023). *Virtualización de redes con el emulador EVE-NG*. Https://Riunet.Upv.Es/Entities/Publication/314bf3e7-3299-47ed-B6c9-0b0c77b40a82.

Costa, L. C., Vieira, A. B., de Britto e Silva, E., Macedo, D. F., Vieira, L. F. M., Vieira, M. A. M., da Rocha Miranda, M., Batista, G. F., Polizer, A. H., Gonçalves, A. V. G. S., Gomes, G., & Correia, L. H. A. (2021). OpenFlow data planes performance evaluation. *Performance Evaluation*, *147*. https://doi.org/10.1016/j.peva.2021.102194

Elmoslemany, M. M., Eldien, A. S. T., & Selim, M. M. (2020, December). Performance Analysis in Software Defined Network Controllers. *Proceedings of ICCES 2020 - 2020 15th International Conference on Computer Engineering and Systems*. https://doi.org/10.1109/ICCES51560.2020.9334577

Golightly, L., Modesti, P., & Chang, V. (2023). Deploying Secure Distributed Systems: Comparative Analysis of GNS3 and SEED Internet Emulator. *Journal of Cybersecurity and Privacy*, *3*, 464–492. https://doi.org/10.3390/jcp3030024

Gómez, D. F. B. (2012). *Openflow: El protocolo del futuro*. Https://Dialnet.Unirioja.Es/Servlet/Articulo?Codigo=4897871.

Gowri, V., & Baranidharan, B. (2025). Adaptive probabilistic neural network based edge data center authentication for secure load balancing in fog computing. *Applied Soft Computing*, *169*. https://doi.org/10.1016/j.asoc.2024.112567

Gures, E., Shayea, I., Ergen, M., Azmi, M. H., & El-Saleh, A. A. (2022). Machine Learning-Based Load Balancing Algorithms in Future Heterogeneous Networks: A Survey. In *IEEE Access* (Vol. 10, pp. 37689–37717). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2022.3161511

Haji, S. H., Zeebaree, S. R. M., Saeed, R. H., Ameen, S. Y., Shukur, H. M., Omar, N., Sadeeq, M. A. M., Ageed, Z. S., Ibrahim, I. M., & Yasin, H. M. (2021). Comparison of Software Defined Networking with Traditional Networking. *Asian Journal of Research in Computer Science*, 1–18. https://doi.org/10.9734/ajrcos/2021/v9i230216

Harahus, M., Čavojský, M., Bugár, G., & Pleva, M. (2023). Interactive Network Learning: An Assessment of EVE-NG Platform in Educational Settings. *Acta Electrotechnica et Informatica*, *23*, 3–9. https://doi.org/10.2478/aei-2023-0011

International Telecommunication Union. (2023). *Recommendation ITU-T Y.1540 (2019) Amd. 2 (03/2023): Internet protocol data communication service – IP packet transfer and availability performance parameters. Amendment 2 – Revised Annex B: Additional search algorithms for IP-based capacity parameters and methods of measurement*. https://handle.itu.int/11.1002/1000/15491

Islam, M. T., Islam, N., & Refat, M. Al. (2020). Node to Node Performance Evaluation through RYU SDN Controller. *Wireless Personal Communications*, *112*, 555–570. https://doi.org/10.1007/s11277-020-07060-4

Khanderao, K. S., & Khedekar, L. S. (2021). Study and systematic Review of Computer Network Simulation with NS-3. In *International Journal of Interdisciplinary Innovative Research &Development*. https://www.tcpdump.org/

Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*, 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Kumar, J., Goomer, R., & Singh, A. K. (2018). Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model for Cloud Datacenters. *Procedia Computer Science*, *125*, 676–682. https://doi.org/10.1016/j.procs.2017.12.087

Kuse, N., & Jaeger, B. (2020). *Network Simulation with ns-3*. https://doi.org/10.2313/NET-2020-11-1_13

Lasso Guamán, D. J., & Puchaicela Condoy, J. R. (2021). *Evaluación del rendimiento de un prototipo SDN (Software Defined Networking) bajo el protocolo OpenFlow utilizando herramientas Open Source en un entorno virtualizado*. Https://Dspace.Ups.Edu.Ec/Handle/123456789/19861.

Latif, Z., Sharif, K., Li, F., Karim, M. M., Biswas, S., & Wang, Y. (2020). A comprehensive survey of interface protocols for software defined networks. In *Journal of Network and Computer Applications* (Vol. 156). Academic Press. https://doi.org/10.1016/j.jnca.2020.102563

Li, Y., Guo, X., Pang, X., Peng, B., Li, X., & Zhang, P. (2020). Performance Analysis of Floodlight and Ryu SDN Controllers under Mininet Simulator. *2020 IEEE/CIC International Conference on Communications in China, ICCC Workshops 2020*, 85–90. https://doi.org/10.1109/ICCCWorkshops49972.2020.9209935

Mahmoudi, M., Barekatain, B., Beheshti, Z., & Quintana, A. A. (2024). MBL-DSDN: a novel load balancing algorithm in distributed software-defined networks based on micro-clustering and B-LSTM methods. *Journal of Supercomputing*, *80*, 20421–20487. https://doi.org/10.1007/s11227-024-06231-5

Márquez Arpi, J. I. (2021). *Análisis y simulación de los algoritmos de scheduling Round Robin, Proportional Fair y Maximum Throughput a nivel de capa física del plano de usuario en redes de nueva generación 5G*. Https://Bibdigital.Epn.Edu.Ec/Handle/15000/21699.

Menédez Arantes Silvia Clara. (2019). *UF1880 - Gestión de redes telemáticas* (5.1). ELEARNING S.L. https://books.google.com.ec/books?hl=es&lr&id=V7hWDwAAQBAJ&oi=fnd&pg=PA1&dq=Modelo+de+Rendimiento+UF1880&ots=9CMA1Z8Ggr&sig=A_gn6n87HetlJ1McBOLgUf7EfWs&redir_esc=y#v=onepage&q=Modelo%20de%20Rendimiento%20UF1880&f=false

Mohammad Nowsin Amin Sheikh, Sawrnali Khatun, Md. Wasim Miah, Sk. Shalauddin Kabir, & Monishanker Halder. (2019). *SDN-Based Approach to Evaluate the Best Controller: Internal Controller NOX and External Controllers POX, ONOS, RYU*. *19*. https://d1wqtxts1xzle7.cloudfront.net/105769357/1cc244ab65c1e0f5b7bd9269d780af9e8580-libre.pdf?1694929852=&response-content-disposition=inline%3B+filename%3DSDN_Based_Approach_to_Evaluate_the_Best.pdf&Expires=1753165672&Signature=YfGfclI0V9y2-P~sIs6QrVEib4U7yGcI3Ch5G~AS06rNkI0dYEvC3IqM2mr80KSxVZKvJ8xLodeWYl89BbSxDLjX1ZN9iLY5RYy3wyXdCJ0y0eMbGPZamcHLm6dkk0yAzcQsv-MW2N6C1AmI5z2MCWJqo34XtT-z2uYRQsJrS53s8PQRWQhOxZ1cpntngpiu2DqAVYQgS9dp2C~pa6w21UciEny-LhOqKQGLhWPWwRn9JkbMnAmYP5P599IaA1iDA7k3R8nsr0vhtB0RnuozeOdcHzv7SBYJWW20ZhGYKCHChNGghje6VEp6lbZQAHAaYrZWlNPCyv7RuLpDZ1WtUg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Mohammed, A. R., Mohammed, S. A., & Shirmohammadi, S. (2019, July). Machine Learning and Deep Learning Based Traffic Classification and Prediction in Software Defined Networking. *2019 IEEE International Symposium on Measurements and Networking, M and N 2019 - Proceedings*. https://doi.org/10.1109/IWMN.2019.8805044

Nejer Haro, D. F. (2025). *DESARROLLO DE UN ALGORITMO DE BALANCEO DE CARGA DINÁMICO DE MÚLTIPLES RUTAS PARA LA REDISTRIBUCIÓN DE FLUJOS EN UNA RED SDN QUE PERMITA MEJORAR EL RENDIMIENTO EN EL ENRUTAMIENTO DE PAQUETES TCP/IP*. https://repositorio.utn.edu.ec/bitstream/123456789/13801/2/04%20RED%20329%20TRABAJO%20GRADO.pdf

Noman, H. M., & Jasim, M. N. (2020). POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator. *IOP Conference Series: Materials Science and Engineering*, *881*. https://doi.org/10.1088/1757-899X/881/1/012102

Omer, Y. A. H., Mustafa, A. B. A., & Abdalla, A. G. (2021, February). Performance Analysis of Round Robin Load Balancing in SDN. *Proceedings of: 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering, ICCCEEE 2020*. https://doi.org/10.1109/ICCCEEE49695.2021.9429662

Permana Arrasyid, F., Stiawan, D., Afifah, N., Heryanto, A., Azhar, I. S. B., Bardadi, A., Yudho Suprapto, B., & Pahendra Anto Saputra, I. (2024). Optimizing Web Server Performance: A Comparative Analysis of Central Manager and Round Robin Load Balancing Algorithms. *ICECOS 2024 - 4th International Conference on Electrical Engineering and Computer Science, Proceeding*, 280–285. https://doi.org/10.1109/ICECOS63900.2024.10791257

Putra, T. D., & Purnomo, R. (2022). Case Study: Improved Round Robin Algorithm. *SinkrOn*, *7*, 950–956. https://doi.org/10.33395/sinkron.v7i3.11530

Rodríguez Herlein, D. R., Talay, C. A., González, C. N., & Marrone, L. A. (2019). *Explorando las redes definidas por software (SDN)*. Https://Sedici.Unlp.Edu.Ar/Handle/10915/103546.

Salmi, T., Kiljander, J., & Pakkala, D. (2020). Stacked boosters network architecture for short-term load forecasting in buildings. *Energies*, *13*. https://doi.org/10.3390/en13092370

Septian, K. A., Istikmal, & Ginting, I. (2021). Analysis of ONOS Clustering Performance on Software Defined Network. *Proceedings of the 2021 IEEE International Conference on Internet of Things and Intelligence Systems, IoTaIS 2021*, 117–122. https://doi.org/10.1109/IoTaIS53735.2021.9628659

Shaik, M., & Kim, S. W. (2025). Security in Wireless Sensor Networks Using OMNET++: Literature Review. In *Sensors* (Vol. 25). Multidisciplinary Digital Publishing Institute (MDPI). https://doi.org/10.3390/s25102972

Sharma, A., Kumar, A., & Kumar, M. (2024). Comparison of Packet Tracer and EVE-NG Tools for Efficient Network Design. *Proceedings of the 18th INDIAcom; 2024 11th International Conference on Computing for Sustainable Global Development, INDIACom 2024*, 917–919. https://doi.org/10.23919/INDIACom61295.2024.10498602

Shi, X., Li, Y., Xie, H., Yang, T., Zhang, L., Liu, P., Zhang, H., & Liang, Z. (2020). An openflow-based load balancing strategy in SDN. *Computers, Materials and Continua*, *62*(1), 385–398. https://doi.org/10.32604/cmc.2020.06418

Simaiya, S., Lilhore, U. K., Sharma, Y. K., Rao, K. B. V. B., Maheswara Rao, V. V. R., Baliyan, A., Bijalwan, A., & Alroobaea, R. (2024). A hybrid cloud load balancing and host utilization prediction method using deep learning and optimization techniques. *Scientific Reports*, *14*(1). https://doi.org/10.1038/s41598-024-51466-0

Singh, I. T., Singh, T. R., & Sinam, T. (2022a). Server Load Balancing with Round Robin Technique in SDN. *2022 International Conference on Decision Aid Sciences and Applications, DASA 2022*, 503–505. https://doi.org/10.1109/DASA54658.2022.9765287

Singh, I. T., Singh, T. R., & Sinam, T. (2022b). Server Load Balancing with Round Robin Technique in SDN. *2022 International Conference on Decision Aid Sciences and Applications, DASA 2022*, 503–505. https://doi.org/10.1109/DASA54658.2022.9765287

Smera, C., & Sandeep, J. (2022). Networks Simulation: Research Based Implementation using Tools and Approaches. *2022 IEEE 3rd Global Conference for Advancement in Technology, GCAT 2022*. https://doi.org/10.1109/GCAT55367.2022.9972119

Tapia Vivero, J. C. (2022). *Testbed para el estudio de la tecnología de redes definidas por software para la carrera de Telecomunicaciones de la Universidad Técnica del Norte*. Https://Repositorio.Utn.Edu.Ec/Handle/123456789/12619.

Tayfour, O. E., Mubarakali, A., Tayfour, A. E., Marsono, M. N., Hassan, E., & Abdelrahman, A. M. (2023). Adapting deep learning-LSTM method using optimized dataset in SDN controller for secure IoT. *Soft Computing*. https://doi.org/10.1007/s00500-023-08348-w

TONKAL, Ö., & POLAT, H. (2021). Traffic Classification and Comparative Analysis with Machine Learning Algorithms in Software Defined Networks. *Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji*, *9*, 71–83. https://doi.org/10.29109/gujsc.869418

Vásquez Figueroa, M. T. (2022). *Análisis comparativo de los algoritmos de balanceo de cargas round robín y least connection de ha-proxy en los sistemas de gestión de base de datos*. Https://Dspace.Utb.Edu.Ec/Handle/49000/15076.

Vishwakarma, S. K. (2022). *Performance Analysis of different Load Balancing Algorithms in SDN Based Data Center Networks*. www.ijert.org

Wilsonprakash, S., & Deepalakshmi, P. (2019, April). Artificial Neural Network Based Load Balancing on Software Defined Networking. *IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing, INCOS 2019*. https://doi.org/10.1109/INCOS45849.2019.8951365

Zakia, U., & Ben Yedder, H. (2017). Dynamic load balancing in SDN-based data center networks. *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2017*, 242–247. https://doi.org/10.1109/IEMCON.2017.8117206

Zhou, Y., & Liu, C. (2024). Server Load Balancing Scheme Based on Weighted-Round Robin Combined with Least Connections Algorithm in SDN. *2024 7th International Conference on Advanced Algorithms and Control Engineering, ICAACE 2024*, 1008–1012. https://doi.org/10.1109/ICAACE61206.2024.10549453

Zhu, L., Karim, M. M., Sharif, K., Li, F., Du, X., & Guizani, M. (2019a). *SDN Controllers: Benchmarking & Performance Evaluation*.

Zhu, L., Karim, M. M., Sharif, K., Li, F., Du, X., & Guizani, M. (2019b). *SDN Controllers: Benchmarking & Performance Evaluation*.

7 Anexos

7.1 Anexo A: Implementación de la Topología de Red (Mininet)

El presente anexo expone el código fuente en lenguaje Python encargado de instanciar la topología de red virtualizada mediante la herramienta Mininet. Este *script* configura la infraestructura base del entorno de simulación experimental, estableciendo tres conmutadores lógicos bajo el protocolo OpenFlow 1.3, cuatro servidores *backend* y doce nodos generadores de carga. Asimismo, el módulo automatiza el despliegue de los servicios emulados y la asignación de las direcciones IP Virtuales (VIPs) necesarias para la evaluación de los controladores SDN.

URL del repositorio:

https://github.com/Franciscoxxxddd/Balanceo-de-carga-con-redes-neuronales-LSTM-con-SDN/blob/f3aee357a662caca6c3e496ea79135caae380112/topo_gns3_full.py

Codigo:

```
#
================================================================================

# Título: topo_gns3_full.py

# Descripción: Despliegue automatizado de topología SDN y servicios en
Mininet

#
================================================================================


from mininet.net import Mininet

from mininet.node import RemoteController, OVSKernelSwitch

from mininet.cli import CLI

from mininet.log import setLogLevel

import os


def start_service(host, name, port, proto='TCP'):
```

```
    """Simula un servicio escuchando en un puerto específico mediante
HTTP Server"""
    svc_dir = f"/tmp/{host.name}_{name}_{port}"
    host.cmd(f"mkdir -p {svc_dir}")
    msg = f"<h1>Respuesta de {name} desde el SERVIDOR
{host.name}</h1>"
    host.cmd(f"echo '{msg}' > {svc_dir}/index.html")

    if proto == 'TCP':
        cmd = f"cd {svc_dir} && python3 -m http.server {port} >
{svc_dir}/http.log 2>&1 &"
        host.cmd(cmd)

def start_web(host):
    """Instancia un servidor web en el puerto 80 por defecto"""
    web_dir = f"/tmp/web_{host.name}"
    host.cmd(f"mkdir -p {web_dir}")
    cmd = (f"cd {web_dir} && "
           f"echo '<h1>Hola desde {host.name} (WEB)</h1>' > index.html
&& "
           f"python3 -m http.server 80 > {web_dir}/http.log 2>&1 &")
    host.cmd(cmd)

def run():
    """Función principal: Creación e interconexión de la topología
SDN"""
    net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)

    print(">>> Creando Topología GNS3 (4 Servicios)...")
    c0 = net.addController('c0', ip='127.0.0.1', port=6633)

    # Instanciación de conmutadores OpenFlow 1.3
    s1 = net.addSwitch('s1', protocols='OpenFlow13') # Capa de
Servidores
```

```
    s3 = net.addSwitch('s3', protocols='OpenFlow13') # Capa Core
    s4 = net.addSwitch('s4', protocols='OpenFlow13') # Capa de
Clientes

    # Instanciación de nodos backend (Servidores) con MAC estáticas
    srv1 = net.addHost('srv1', ip='10.0.0.1', mac='00:00:00:00:00:01')
    srv2 = net.addHost('srv2', ip='10.0.0.2', mac='00:00:00:00:00:02')
    srv3 = net.addHost('srv3', ip='10.0.0.3', mac='00:00:00:00:00:03')
    srv4 = net.addHost('srv4', ip='10.0.0.4', mac='00:00:00:00:00:04')

    # Instanciación de nodos de carga (Clientes)
    cli1 = net.addHost('cli1', ip='10.0.0.11')
    cli2 = net.addHost('cli2', ip='10.0.0.12')
    cli3 = net.addHost('cli3', ip='10.0.0.13')
    cli4 = net.addHost('cli4', ip='10.0.0.14')
    cli5 = net.addHost('cli5', ip='10.0.0.15')
    cli6 = net.addHost('cli6', ip='10.0.0.16')
    cli7 = net.addHost('cli7', ip='10.0.0.17')
    cli8 = net.addHost('cli8', ip='10.0.0.18')
    cli9 = net.addHost('cli9', ip='10.0.0.19')
    cli10 = net.addHost('cli10', ip='10.0.0.20')
    cli11 = net.addHost('cli11', ip='10.0.0.21')
    cli12 = net.addHost('cli12', ip='10.0.0.22')

    # Establecimiento de enlaces físicos virtuales
    for h in [srv1, srv2, srv3, srv4]:
        net.addLink(s1, h)

    for h in [cli1, cli2, cli3, cli4, cli5, cli6, cli7, cli8, cli9,
cli10, cli11, cli12]:
        net.addLink(s4, h)

    net.addLink(s1, s3)
```

```
    net.addLink(s4, s3)

    net.start()

    print(">>>  INICIANDO 4 SERVICIOS EN CADA NODO...")
    for srv in [srv1, srv2, srv3, srv4]:
        start_web(srv)
        start_service(srv, "STREAMING", 8080)
        start_service(srv, "DATABASE", 3306)
        start_service(srv, "VOIP", 5060)

    print("\n" + "="*60)
    print(" RESUMEN DE LA TOPOLOGÍA MININET")
    print(" VIP WEB:       10.0.0.100 (Port 80)")
    print(" VIP STREAMING: 10.0.0.101 (Port 8080)")
    print(" VIP DATABASE:  10.0.0.200 (Port 3306)")
    print(" VIP VOIP:      10.0.0.201 (Port 5060)")
    print("="*60 + "\n")

    CLI(net)
    net.stop()

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

7.2 Anexo B. Lógica Base del Controlador Estático (Round Robin)

El Anexo B presenta el código fuente del controlador SDN estático desarrollado sobre el *framework* Ryu, el cual implementa el algoritmo tradicional Round Robin como línea base experimental de la investigación. Este *script* opera bajo el protocolo OpenFlow 1.3 y ejecuta la asignación secuencial y cíclica de las peticiones de red hacia los cuatro servidores *backend*. Además de la distribución de carga, el módulo gestiona de forma autónoma la resolución ARP

de las direcciones IP Virtuales (VIP), asegura la persistencia de las sesiones (*sticky sessions*) para evitar la disrupción de las conexiones TCP, y exporta la telemetría operativa en formato JSON para su posterior análisis.

URL del Repositorio:

https://github.com/Franciscoxxxddd/Balanceo-de-carga-con-redes-neuronales-LSTM-con-SDN/blob/f3aee357a662caca6c3e496ea79135caae380112/ryu_service_rr.py

Codigo:

```
#
================================================================================
========

# Título: ryu_service_rr.py

# Descripción: Controlador SDN Ryu implementando balanceo de carga
estático (Round Robin)

#
================================================================================
========


from ryu.base import app_manager

from ryu.controller import ofp_event

from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER,
set_ev_cls

from ryu.ofproto import ofproto_v1_3

from ryu.lib.packet import packet, ethernet, arp, ipv4, tcp, udp,
ether_types

import json, time, random


class GNS3RoundRobinLoadBalancer(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]


    def __init__(self, *args, **kwargs):

        super(GNS3RoundRobinLoadBalancer, self).__init__(*args,
**kwargs)

        self.logger.info(">>> [INIT] Iniciando Controlador SDN (Round
Robin)...")


        # Variable de estado para el ciclo secuencial del Round Robin
```

```
        self.rr_counter = 0
        self.DASHBOARD_FILE = 'estado_red.json'
        self.last_write = 0

        # Tabla de Sesiones (Sticky Sessions) para mantener conexiones
TCP vivas
        self.sticky_sessions = {}

        # Definición de IPs Virtuales (VIPs) orientadas a servicios
        self.SERVICES = {
            '10.0.0.100': {'name': 'WEB', 'mac': '00:00:00:00:FE:80',
'port': 80},
            '10.0.0.101': {'name': 'STREAMING', 'mac':
'00:00:00:00:FE:81', 'port': 8080},
            '10.0.0.200': {'name': 'DATABASE', 'mac':
'00:00:00:00:FE:33', 'port': 3306},
            '10.0.0.201': {'name': 'VOIP', 'mac': '00:00:00:00:FE:50',
'port': 5060}
        }

        # Definición de la infraestructura física (Servidores Backend)
        self.servers = {
            'srv1': {'ip': '10.0.0.1', 'mac': '00:00:00:00:00:01'},
            'srv2': {'ip': '10.0.0.2', 'mac': '00:00:00:00:00:02'},
            'srv3': {'ip': '10.0.0.3', 'mac': '00:00:00:00:00:03'},
            'srv4': {'ip': '10.0.0.4', 'mac': '00:00:00:00:00:04'}
        }

    def get_round_robin_server(self):
        """Implementación matemática del algoritmo secuencial
estático"""
        server_keys = list(self.servers.keys())
        # Selección mediante módulo aritmético para garantizar el
ciclo
```

```
        chosen = server_keys[self.rr_counter % len(server_keys)]
        self.rr_counter += 1
        return chosen

    def _balancear(self, datapath, msg, pkt, ip_pkt, in_port, svc):
        """Función principal de conmutación de flujos hacia el
backend"""
        # Creación de llave de sesión ligada a la IP origen y al
servicio
        session_key = (ip_pkt.src, svc['name'])

        # Verificación de persistencia: si el cliente ya tiene un
servidor asignado
        if session_key in self.sticky_sessions:
            chosen = self.sticky_sessions[session_key]
        else:
            # Asignación de un nuevo servidor mediante Round Robin
            metrics = self._get_live_metrics()
            chosen = self.get_round_robin_server()
            self.sticky_sessions[session_key] = chosen
            self.logger.info(f" [{svc['name']}] -> RR asignó {chosen}
al cliente {ip_pkt.src}")
            self.export_dashboard(ip_pkt.src, svc['name'], chosen,
metrics)

        # Extracción de parámetros del servidor seleccionado
        srv_data = self.servers[chosen]
        parser = datapath.ofproto_parser

        # Modificación de las cabeceras (NAT destino)
        actions = [
            parser.OFPActionSetField(eth_dst=srv_data['mac']),
            parser.OFPActionSetField(ipv4_dst=srv_data['ip']),
            parser.OFPActionOutput(datapath.ofproto.OFPP_FLOOD)
```

```
        ]


        # Inyección de la regla en el plano de datos
        out = parser.OFPPacketOut(datapath=datapath,
buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions,
data=msg.data)
        datapath.send_msg(out)
```

7.3 Anexo C. Lógica de Inferencia del Controlador Inteligente (LSTM)

El presente anexo detalla el código fuente del plano de control predictivo desarrollado sobre el *framework* Ryu, el cual integra una arquitectura de red neuronal de Memoria a Largo y Corto Plazo Bidireccional (Bi-LSTM). Este módulo es el encargado de romper con la lógica secuencial tradicional, capturando la telemetría de la red en tiempo real para preprocesar secuencias temporales en tensores tridimensionales. Mediante la ejecución de inferencias sobre un modelo preentrenado, el controlador predice el estado computacional futuro de los nodos *backend* y emite dinámicamente mensajes OpenFlow (Flow-Mod) para ejecutar conmutaciones proactivas, mitigando la congestión y garantizando la alta disponibilidad del clúster.

URL del Repositorio:

https://github.com/Franciscoxxxddd/Balanceo-de-carga-con-redes-neuronales-LSTM-con-SDN/blob/f3aee357a662caca6c3e496ea79135caae380112/ryu_service_ai.py

Codigo:

```
#
================================================================================

# Título: ryu_service_ai.py (Fragmento principal de inferencia)

# Descripción: Controlador SDN con toma de decisiones predictiva
mediante Bi-LSTM

#
================================================================================


from ryu.base import app_manager

from ryu.controller import ofp_event

from ryu.controller.handler import MAIN_DISPATCHER, set_ev_cls

from ryu.ofproto import ofproto_v1_3

import numpy as np
```

```
import tensorflow as tf
from tensorflow.keras.models import load_model


class GNS3IntelligentLoadBalancer(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]


    def __init__(self, *args, **kwargs):
        super(GNS3IntelligentLoadBalancer, self).__init__(*args,
**kwargs)
        self.logger.info(">>> [INIT] Iniciando Controlador SDN
Inteligente (Bi-LSTM)...")


        # Carga del modelo neuronal preentrenado en memoria
        try:
            self.ai_model = load_model('bilstm_sdn_model.h5')
            self.logger.info(">>> [AI] Modelo Bi-LSTM cargado
exitosamente.")
        except Exception as e:
            self.logger.error(f"Error al cargar el modelo AI: {e}")


        # Diccionario de servidores backend y umbral crítico de CPU
        self.servers = ['srv1', 'srv2', 'srv3', 'srv4']
        self.CRITICAL_CPU_THRESHOLD = 90.0


    def predict_optimal_server(self, current_telemetry):
        """
        Transforma la telemetría en tensores y ejecuta la inferencia
de red neuronal
        para predecir el estado de saturación y seleccionar el nodo
óptimo.
        """
        # Preprocesamiento: Conversión de datos planos a matriz
temporal para LSTM
        # Forma esperada del tensor: (samples, timesteps, features)
```

```
        input_tensor = np.array(current_telemetry).reshape(1, 5,
len(self.servers))


        # Inferencia matemática: Predicción de carga futura
        predicted_loads = self.ai_model.predict(input_tensor,
verbose=0)[0]


        optimal_server = None
        lowest_load = float('inf')


        # Evaluación de mitigación proactiva: Buscar el servidor con
menor carga predicha
        for idx, srv in enumerate(self.servers):
            future_load = predicted_loads[idx]


            # Penalización algorítmica si el nodo excede el umbral
crítico
            if future_load >= self.CRITICAL_CPU_THRESHOLD:
                self.logger.warning(f"⚠️ [ALERTA] {srv} superará el
{self.CRITICAL_CPU_THRESHOLD}% de CPU.")
                continue


            if future_load < lowest_load:
                lowest_load = future_load
                optimal_server = srv


        # Fallback de emergencia si todos los servidores predicen
saturación
        if optimal_server is None:
            optimal_server = self.servers[np.argmin(predicted_loads)]


        return optimal_server
```

```
    def _balancear_inteligente(self, datapath, msg, pkt, in_port,
svc):

        """Intervención en el plano de datos basada en la decisión de
la IA"""

        # Recolección de la ventana de tiempo de telemetría
(Timesteps)

        telemetry_window = self._get_recent_telemetry_window()


        # Decisión inteligente (Divergencia del modelo secuencial)

        chosen_server = self.predict_optimal_server(telemetry_window)


        self.logger.info(f" [IA] Mitigación proactiva: Tráfico
desviado hacia {chosen_server}")


        # Generación de la regla OpenFlow (Flow-Mod) para aplicar la
predicción

        self._apply_flow_rule(datapath, msg, in_port, chosen_server,
svc)
```

7.4 Anexo D. Scripts de Automatización de Pruebas y Generación de Tráfico

El presente anexo expone un fragmento del módulo común de pruebas desarrollado en lenguaje Python (test_common.py), el cual estandariza la ejecución de los escenarios comparativos entre los controladores SDN. La implementación de este *script* garantiza la repetibilidad del diseño experimental mediante la fijación de semillas de aleatoriedad (Seed) y la automatización en la inyección de tráfico. Adicionalmente, el código evidencia la programación algorítmica utilizada para la captura y el cálculo de las métricas de rendimiento exigidas por los estándares internacionales ITU-T Y.1540 (IPLR, IPTD, IPDV) y RFC 2544 (Throughput, Back-to-Back), extrayendo los datos directamente desde los nodos virtualizados en Mininet.

URL del Repositorio:

https://github.com/Franciscoxxxddd/Balanceo-de-carga-con-redes-neuronales-LSTM-con-SDN/blob/f3aee357a662caca6c3e496ea79135caae380112/pruebas/test_common.py

Código:

```
#!/usr/bin/env python3

#
================================================================================

# Título: test_common.py

# Descripción: Módulo común para las 10 pruebas comparativas SDN (RR
vs IA)
```

```
# Estándares: Métricas ITU-T Y.1540 (IPLR, IPTD, IPDV) + RFC 2544
#
================================================================================


import os, sys, time, json, re, random, statistics, subprocess,
threading
from mininet.net import Mininet
from mininet.link import TCLink


# --- CONSTANTES GLOBALES Y REPRODUCIBILIDAD ---
SEED = 42                               # Semilla fija para
reproducibilidad
RNG = random.Random(SEED)               # Generador determinista
compartido
BW_TRUNK = 10                           # Ancho de banda troncal (Mbps)
BW_ACCESS = 100                         # Ancho de banda de acceso (Mbps)
QUEUE_SIZE = 50                         # Tamaño de cola en switches core
MIN_PACKETS = 70                        # Mínimo de paquetes por flujo


# --- MÉTRICAS ESTÁNDAR ITU-T Y.1540 ---


def medir_iptd(src, dst_ip, count=20):
    """
    IP Packet Transfer Delay (ITU-T Y.1540)
    Extrae la latencia de transferencia mediante peticiones ICMP.
    Retorna: diccionario con métricas min/avg/max/mdev y pérdida.
    """
    result = src.cmd(f"ping -c {count} -W 2 -i 0.1 {dst_ip}")
    m = re.search(r'rtt min/avg/max/mdev =
([\d.]+)/([\d.]+)/([\d.]+)/([\d.]+)', result)
    if m:
        loss_m = re.search(r'([\d.]+)% packet loss', result)
        loss = float(loss_m.group(1)) if loss_m else 0.0
```

```
        return {
            'min': float(m.group(1)),
            'avg': float(m.group(2)),
            'max': float(m.group(3)),
            'mdev': float(m.group(4)),
            'loss': loss
        }
    return {'min': 0, 'avg': 0, 'max': 0, 'mdev': 0, 'loss': 100.0}


def medir_iplr(src, dst_ip, count=50):
    """
    IP Packet Loss Ratio (ITU-T Y.1540)
    Mide el porcentaje exacto de paquetes perdidos con precisión
decimal.
    """
    result = src.cmd(f"ping -c {count} -W 2 -i 0.05 {dst_ip}")
    tx_m = re.search(r'(\d+) packets transmitted', result)
    rx_m = re.search(r'(\d+) received', result)
    loss_m = re.search(r'([\d.]+)% packet loss', result)

    enviados = int(tx_m.group(1)) if tx_m else count
    recibidos = int(rx_m.group(1)) if rx_m else 0
    perdidos = enviados - recibidos
    porcentaje = float(loss_m.group(1)) if loss_m else (
        (perdidos / enviados * 100) if enviados > 0 else 100.0)

    return {
        'enviados': enviados,
        'recibidos': recibidos,
        'perdidos': perdidos,
        'porcentaje': round(porcentaje, 2)
    }
```

```
# --- MÉTRICAS ESTÁNDAR RFC 2544 ---


def medir_throughput_servidor(server_host, client_host, port=5001,
duration=10):
    """
    Throughput (RFC 2544) - Medido desde la perspectiva del servidor.
    Retorna el ancho de banda efectivo en Mbps.
    """
    server_host.cmd(f"iperf -s -p {port} &")
    time.sleep(1)
    result = client_host.cmd(f"iperf -c {server_host.IP()} -p {port} -
t {duration} -f m")
    server_host.cmd(f"kill %iperf 2>/dev/null")


    m = re.search(r'([\d.]+)\s+Mbits/sec', result)
    return float(m.group(1)) if m else 0.0


# [ ... Resto del código omitido por brevedad en la impresión ... ]
# El script completo incluye funciones adicionales para IPDV,
# System Recovery y cálculos de equidad de servidores.
```

7.5 Anexo E. Estructura de Telemetría y Registro de Decisiones (JSON)

El presente anexo detalla la arquitectura de los archivos de telemetría utilizados durante la fase experimental para la recolección de datos y el análisis comparativo. La información se almacena bajo el formato de notación de objetos de JavaScript (JSON), permitiendo la estructuración jerárquica de métricas de red bajo los estándares ITU-T Y.1540 y RFC 2544. Estos archivos actúan como la base de datos operativa del controlador SDN, permitiendo el contraste entre la lógica determinista del algoritmo *Round Robin* y la inferencia proactiva del modelo Bi-LSTM.

URL del Repositorio:

https://github.com/Franciscoxxxddd/Balanceo-de-carga-con-redes-neuronales-LSTM-con-SDN/tree/f3aee357a662caca6c3e496ea79135caae380112/pruebas/reportes

- Estructura del Registro de Asignación Comparativa:


```
{
  "titulo": "Reporte de Asignación Comparativa - Servidor por Petición",
  "pruebas": [
    {
      "prueba": 1,
      "nombre": "Tráfico Base",
      "asignaciones": [
        {
          "peticion": 1,
          "servidor_RR": "srv1",
          "servidor_IA": "srv2",
          "coincide": false
        }
      ]
    }
  ]
}
```


- Estructura del Registro de Conmutación por Saturación (IA):


```
{
  "titulo": "Reporte de Conmutación por Saturación - IA-LSTM",
  "eventos_por_prueba": [
    {
      "prueba": 3,
      "eventos": [
        {
          "timestamp": 1777966220.87,
          "servidor_original": "srv2",
          "cpu_detectada": 100.0,
          "nuevo_servidor": "srv4",
```

```
          "fue_desviado": true
        }
      ]
    }
  ]
}
```


- Estructura de Métricas Globales de Rendimiento:


```
{
  "titulo": "Reporte de Métricas Globales - RR vs IA",
  "pruebas": [
    {
      "prueba": 1,
      "RR": { "iplr_percent": 0.0, "iptd_avg_ms": 1.07 },
      "IA": { "iplr_percent": 0.0, "iptd_avg_ms": 0.94 },
      "diferencia": { "iptd_avg_ms": -0.13 }
    }
  ]
}
```


7.6 Anexo F. Interfaz de Monitoreo y Dashboard de Control SDN

Este anexo documenta la interfaz gráfica desarrollada mediante el *framework* Streamlit (dashboard_tesis.py), diseñada para el monitoreo en tiempo real y la evaluación comparativa de las métricas de red. El tablero permite visualizar de forma interactiva la topología de la red, la utilización de recursos de los servidores (CPU/RAM) y el historial de decisiones tomadas por los controladores Round Robin y el modelo Bi-LSTM. La herramienta se integra directamente con los registros JSON del sistema, facilitando la auditoría de los eventos de conmutación proactiva y la comparación técnica bajo los estándares ITU-T y RFC 2544, brindando al investigador una capacidad de diagnóstico inmediato sobre el comportamiento dinámico de los flujos de tráfico.

URL del Repositorio:

https://github.com/Franciscoxxxddd/Balanceo-de-carga-con-redes-neuronales-LSTM-con-SDN/blob/f3aee357a662caca6c3e496ea79135caae380112/dashboard_tesis.py

Codigo:

```
#
================================================================================
========

# Título: dashboard_tesis.py (Fragmento de visualización)

# Descripción: Funciones de graficación y renderizado de métricas en
Streamlit

#
================================================================================
========


import streamlit as st

import plotly.graph_objects as go


def crear_grafico_barras(categorias, valores_dict, titulo, unidad='%',
height=350):

    """Crea un gráfico de barras agrupadas para comparar RR vs IA"""

    colores = ['#1E90FF', '#32CD32'] # Azul (RR) vs Verde (IA)

    fig = go.Figure()


    for i, (nombre, vals) in enumerate(valores_dict.items()):

        color = colores[i % len(colores)]

        fig.add_trace(go.Bar(

            name=nombre,

            x=categorias,

            y=vals,

            marker_color=color,

            text=[f'{v:.1f}{unidad}' for v in vals],

            textposition='outside',

            textfont=dict(size=11, color=color)

        ))


    fig.update_layout(

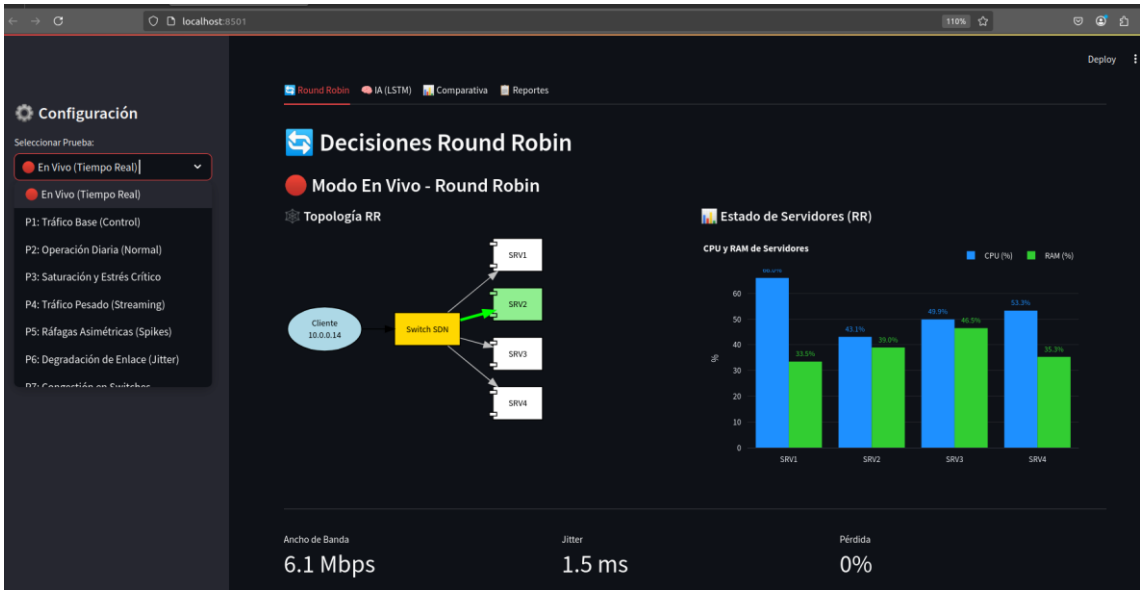
        title=dict(text=titulo, font=dict(size=14)),
```

```
        barmode='group',
        height=height,
        margin=dict(l=40, r=40, t=50, b=40),
        yaxis=dict(title=unidad),
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)'
    )
    return fig

def crear_gauge_chart(valor, titulo, max_val=100, color='#1E90FF'):
    """Indicador visual tipo velocímetro para métricas de pérdida
(IPLR)"""
    fig = go.Figure(go.Indicator(
        mode = "gauge+number",
        value = valor,
        title = {'text': titulo, 'font': {'size': 14}},
        gauge = {
            'axis': {'range': [0, max_val]},
            'bar': {'color': color},
            'steps': [
                {'range': [0, 5], 'color': 'rgba(0, 255, 0, 0.1)'},
                {'range': [5, 15], 'color': 'rgba(255, 255, 0, 0.1)'},
                {'range': [15, 100], 'color': 'rgba(255, 0, 0, 0.1)'}
            ]
        }
    ))
    return fig

# [ ... ] Este módulo integra estas funciones dentro de los tabs del
dashboard
# para renderizar dinámicamente los reportes JSON cargados.
```

Capturas de Pantalla del Dashboard:






localhost:8501
Deploy
Configuración
Seleccionar Prueba:
En Vivo (Tiempo Real)
Limpiar Todos los Historiales
Comparativa En Vivo
Controlador Activo
CPU
0.0%
RAM
0.0%
Servidor Elegido
SRV2
Tráfico
BAJO
Topología en Tiempo Real
Cliente
10.0.0.14
Switch SDN
SRV1
SRV3
SRV4
CPU y RAM de Servidores
CPU y RAM de Servidores
CPU (%)
RAM (%)
SRV1
SRV2
SRV3
SRV4


Deploy
Configuración
Seleccionar Prueba:
En Vivo (Tiempo Real)
Limpiar Todos los Historiales
Ejecuta python3 pruebas/generar_reportes.py después de las 20 pruebas para generar estos reportes.
1 Métricas Globales (ITU-T Y.1540 + RFC 2544)
Prueba | Nombre | IPLR RR | IPLR IA | IPTD RR | IPTD IA | IPDV RR | IPDV IA
P1 | Tráfico Base | 0.0% | 0.0% | 1.07ms | 0.94ms | 1.65ms | 1.83ms
P2 | Operación Diaria | 0.0% | 0.0% | 1.07ms | 1.07ms | 1.66ms | 3.21ms
P3 | Estrés Crítico | 0.12% | 0.0% | 15.18ms | 15.47ms | 2.86ms | 2.93ms
P4 | Streaming | 0.0% | 0.0% | 1.53ms | 1.43ms | 3.02ms | 2.85ms
P5 | Ráfagas | 5.95% | 5.83% | 26.59ms | 30.23ms | 7.26ms | 5.7ms
P6 | Jitter | 0.0% | 0.0% | 0.95ms | 1.23ms | 2.49ms | 2.42ms
P7 | Congestión | 4.88% | 5.0% | 0.17ms | 0.16ms | 0.08ms | 0.08ms
P8 | Tráfico Mixto | 0.0% | 0.0% | 1.44ms | 1.07ms | 2.86ms | 1.91ms
P9 | Caída Servidor | 2.62% | 0.0% | 135.63ms | 10.39ms | 28.93ms | 2.88ms
P10 | Resiliencia | 0.0% | 0.0% | 1.2ms | 1.11ms | 2.27ms | 2.28ms
2 Asignación Comparativa (RR vs IA)
Prueba | Nombre | Decisiones Comparadas | Coincidencias | % Coincidencia
P1 | Tráfico Base | 27 | 4 | 14.8%
P2 | Operación Diaria | 25 | 3 | 12.0%

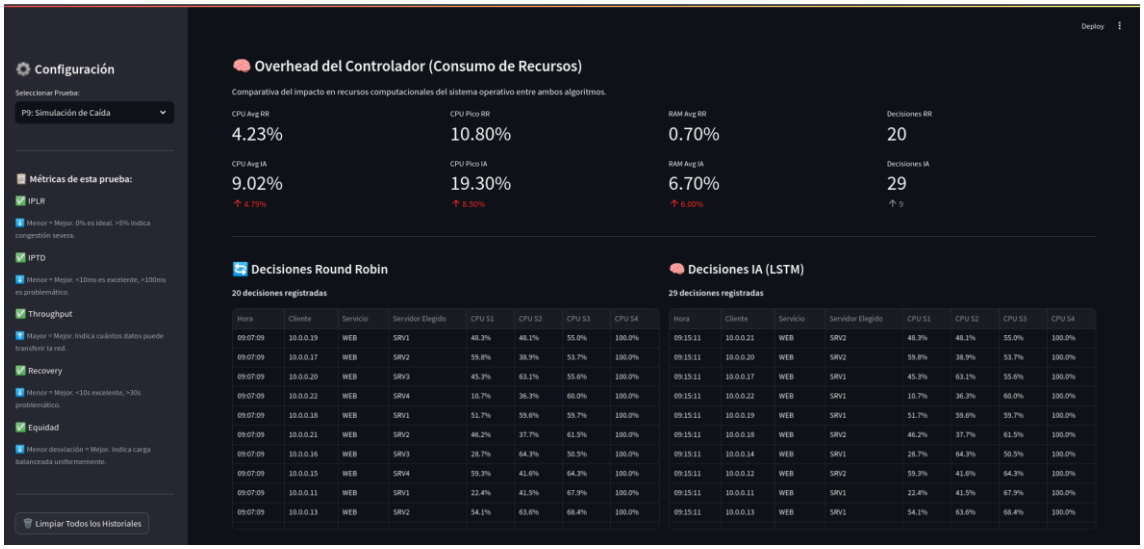

Deploy
Configuración
Seleccionar Prueba:
En Vivo (Tiempo Real)
Limpiar Todos los Historiales
3 Conmutación por Saturación (Solo IA)
Total Eventos de Evicción
268
P1: Tráfico Base (7 eventos)
P2: Operación Diaria (5 eventos)
P3: Estrés Crítico (5 eventos)
P4: Streaming (6 eventos)
P5: Ráfagas (3 eventos)
P6: Jitter (1 eventos)
P7: Congestión (3 eventos)
P8: Tráfico Mixto (6 eventos)
P9: Caída Servidor (38 eventos)


Deploy
Configuración
Seleccionar Prueba:
P10: Resiliencia (15 min)
Métricas de esta prueba:
IPLR
Menor = Mejor. 0% es ideal. >5% indica congestión severa.
IPTD
Menor = Mejor. <10ms es excelente, >100ms es problemático.
Throughput
Mayor = Mejor. Indica cuántos datos puede transferir la red.
Recovery
Menor = Mejor. <10s excelente, >30s problemático.
Equidad temporal
Decisiones Round Robin
Controlador Round Robin
CPU Avg
19.27%
CPU Max
23.60%
RAM Avg
0.70%
RAM Max
0.70%
Tiempo Respuesta Avg
55230.90 ms
Total Decisiones
20
Métricas de la Prueba
Latencia
Host
Avg (ms)
Min (ms)
Max (ms)
cli3 1.40 0.10 23.59
cli4 0.77 0.10 10.98
cli5 1.21 0.08 19.82
cli6 1.23 0.08 20.84
cli7 0.74 0.07 9.64
cli8 1.22 0.09 19.93


Deploy
Configuración
Seleccionar Prueba:
P10: Resiliencia (15 min)
Métricas de esta prueba:
IPLR
Menor = Mejor. 0% es ideal. >5% indica congestión severa.
IPTD
Menor = Mejor. <10ms es excelente, >100ms es problemático.
Throughput
Mayor = Mejor. Indica cuántos datos puede transferir la red.
Recovery
Menor = Mejor. <10s excelente, >30s problemático.
Equidad temporal
Decisiones IA (LSTM)
Controlador IA (LSTM)
CPU Avg
21.71%
CPU Max
27.60%
RAM Avg
6.70%
RAM Max
6.70%
Tiempo Respuesta Avg
4147.35 ms
Total Decisiones
254
Métricas de la Prueba
Latencia
Host
Avg (ms)
Min (ms)
Max (ms)
cli1 1.01 0.08 15.64
cli2 0.71 0.10 9.23
cli3 1.52 0.07 26.70
cli4 1.13 0.10 19.26
cli5 0.94 0.09 13.50
cli6 1.06 0.10 16.71
cli7 0.77 0.10 11.45


Deploy
Configuración
Seleccionar Prueba:
P10: Resiliencia (15 min)
Métricas de esta prueba:
IPLR
Menor = Mejor. 0% es ideal. >5% indica congestión severa.
IPTD
Menor = Mejor. <10ms es excelente, >100ms es problemático.
Throughput
Mayor = Mejor. Indica cuántos datos puede transferir la red.
Recovery
Menor = Mejor. <10s excelente, >30s problemático.
Equidad temporal
Latencia (IPTD)
37.3%
37.3%
Pérdida (IPLR)
0.0%
0.0%
Jitter (IPDV)
-0.4%
-0.4%
Throughput
1.3%
1.3%
Estándar ITU-T Y.1540 (Calidad de Experiencia)
1. IP Packet Transfer Delay (IPTD)
Comprende el retardo de transmisión más el tiempo de respuesta/inferencia del controlador SDN.
Latencia Extremo a Extremo (IPTD)
Round Robin (Red)
IA LSTM (Red)
ms
1.8
1.6
1.4
1.2
1
0.8
cli1
cli10
cli11
cli12
cli2
cli3
cli4
cli5
cli6
cli7
cli8
cli9
Tiempo de Respuesta RR
55230.90 ms
Tiempo de Inferencia IA
4147.35 ms


Configuración
Seleccionar Prueba:
P9: Simulación de Caída
Métricas de esta prueba:
IPLR
Menor = Mejor. 0% es ideal. >5% indica congestión severa.
IPTD
Menor = Mejor. <10ms es excelente, >100ms es problemático.
Throughput
Mayor = Mejor. Indica cuántos datos puede transferir la red.
Recovery
Menor = Mejor. <10s excelente, >30s problemático.
Equidad
Menor desviación = Mejor. Indica carga balanceada uniformemente.
Limpiar Todos los Historiales
Deploy
2. IP Packet Loss Ratio (IPLR)
Pérdida Round Robin (%)
2.6
Pérdida IA LSTM (%)
0
3. IP Packet Delay Variation (IPDV) - Streaming 8080 / VoIP 5060
Menor jitter = Mejor. <5ms aceptable para VoIP, <10ms para streaming.
Host
Streaming RR (ms)
Streaming IA (ms)
VoIP RR (ms)
VoIP IA (ms)
cli1
7.17
12.55
12.00
10.90
cli2
11.75
13.76
112.75
8.22
cli3
11.34
13.30
7.65
9.45
cli4
8.53
13.23
168.53
12.42
Jitter RR (Streaming+VoIP)
Frecuencia
ms
Jitter IA (Streaming+VoIP)
Frecuencia
ms


Configuración
Seleccionar Prueba:
P9: Simulación de Caída
Métricas de esta prueba:
IPLR
Menor = Mejor. 0% es ideal. >5% indica congestión severa.
IPTD
Menor = Mejor. <10ms es excelente, >100ms es problemático.
Throughput
Mayor = Mejor. Indica cuántos datos puede transferir la red.
Recovery
Menor = Mejor. <10s excelente, >30s problemático.
Equidad
Menor desviación = Mejor. Indica carga balanceada uniformemente.
Limpiar Todos los Historiales
Deploy
Overhead del Controlador (Consumo de Recursos)
Comparativa del impacto en recursos computacionales del sistema operativo entre ambos algoritmos.
CPU Avg RR
4.23%
CPU Pico RR
10.80%
RAM Avg RR
0.70%
Decisiones RR
20
CPU Avg IA
9.02%
↑ 4.79%
CPU Pico IA
19.30%
↑ 8.50%
RAM Avg IA
6.70%
↑ 6.00%
Decisiones IA
29
↑ 9
Decisiones Round Robin
20 decisiones registradas
Hora
Cliente
Servicio
Servidor Elegido
CPU S1
CPU S2
CPU S3
CPU S4
09:07:09
10.0.0.19
WEB
SRV1
48.3%
48.1%
55.0%
100.0%
09:07:09
10.0.0.17
WEB
SRV2
59.8%
38.9%
53.7%
100.0%
09:07:09
10.0.0.20
WEB
SRV3
45.3%
63.1%
55.6%
100.0%
09:07:09
10.0.0.22
WEB
SRV4
10.7%
36.3%
60.0%
100.0%
09:07:09
10.0.0.18
WEB
SRV1
51.7%
59.6%
59.7%
100.0%
09:07:09
10.0.0.21
WEB
SRV2
46.2%
37.7%
61.5%
100.0%
09:07:09
10.0.0.16
WEB
SRV3
28.7%
64.3%
50.5%
100.0%
09:07:09
10.0.0.15
WEB
SRV4
59.3%
41.6%
64.3%
100.0%
09:07:09
10.0.0.11
WEB
SRV1
22.4%
41.5%
67.9%
100.0%
09:07:09
10.0.0.13
WEB
SRV2
54.1%
63.6%
68.4%
100.0%
Decisiones IA (LSTM)
29 decisiones registradas
09:15:11
10.0.0.21
WEB
SRV2
48.3%
48.1%
55.0%
100.0%
09:15:11
10.0.0.20
WEB
SRV2
59.8%
38.9%
53.7%
100.0%
09:15:11
10.0.0.17
WEB
SRV1
45.3%
63.1%
55.6%
100.0%
09:15:11
10.0.0.22
WEB
SRV1
10.7%
36.3%
60.0%
100.0%
09:15:11
10.0.0.19
WEB
SRV1
51.7%
59.6%
59.7%
100.0%
09:15:11
10.0.0.18
WEB
SRV2
46.2%
37.7%
61.5%
100.0%
09:15:11
10.0.0.14
WEB
SRV1
28.7%
64.3%
50.5%
100.0%
09:15:11
10.0.0.12
WEB
SRV2
59.3%
41.6%
64.3%
100.0%
09:15:11
10.0.0.11
WEB
SRV1
22.4%
41.5%
67.9%
100.0%
09:15:11
10.0.0.13
WEB
SRV1
54.1%
63.6%
68.4%
100.0%