UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE
ESCUELA POLITÉCNICA SUPERIOR DE ELCHE
GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Implementación SDN de Arquitecturas Fat-Tree Multidimensional

TRABAJO FIN DE GRADO

Junio - 2026

AUTOR: Fernando Tomás Gámez Cartagena
DIRECTOR/ES: Salvador Alcaraz Carrasco
Pedro Juan Roig Roig

AGRADECIMIENTOS

En primer lugar, quiero expresar mi agradecimiento a mis directores, Salvador Alcaraz Carrasco y Pedro Juan Roig Roig, por su orientación, disponibilidad y acompañamiento durante el desarrollo de este Trabajo Fin de Grado. Sus indicaciones han sido fundamentales para encauzar el proyecto, mantener una línea de trabajo rigurosa y transformar una propuesta inicial amplia en una memoria técnica coherente y defendible.

También quiero agradecer a la Universidad Miguel Hernández de Elche y a la Escuela Politécnica Superior de Elche la formación recibida durante estos años. Este trabajo representa no solo el cierre de una etapa académica, sino también el resultado de un proceso de aprendizaje acumulado en asignaturas, prácticas, proyectos y experiencias que han contribuido a mi desarrollo como estudiante de Ingeniería de Tecnologías de Telecomunicación.

De manera especial, quiero dar las gracias a mi familia, por su apoyo constante, su paciencia y su comprensión durante todo este camino. A mi mujer, Sari, porque sin su sonrisa nada de esto tendría sentido. A mi hijo, Leonardo, porque sin sus abrazos me habría derrumbado más de una vez. A mi padre, Tomás, por su maravilloso ejemplo, sé que estás empujando desde allá arriba. A mi madre, Aurora, porque sin su luz seguiría pensando que la noche nunca acaba. En los momentos de mayor carga de trabajo, su apoyo ha sido esencial para seguir avanzando y no perder de vista el objetivo final.

Asimismo, quiero agradecer a aquellas personas que, de una forma u otra, me han acompañado durante esta etapa: compañeros, profesores y amigos como Fran, Tomás y Amine, con quienes he compartido dudas, consejos, conversaciones y momentos de esfuerzo. Cada aportación, por pequeña que pareciera, ha formado parte del recorrido que culmina en este trabajo.

Finalmente, quiero dedicar unas palabras al esfuerzo personal que ha supuesto este proyecto. La realización de este TFG ha requerido constancia, aprendizaje autónomo, capacidad de adaptación y muchas horas de prueba, error y revisión. Por ello, cerrar esta memoria supone también reconocer el valor de todo ese proceso y de la perseverancia necesaria para llegar hasta aquí.

"Fija tu rumbo a una estrella y podrás navegar a través de cualquier tormenta."
Inspirada en una reflexión de Leonardo da Vinci

RESUMEN

Este Trabajo Fin de Grado aborda el diseño, implementación y evaluación de un banco de pruebas SDN orientado al estudio de topologías de centro de datos, tomando como arquitectura principal una topología Fat-Tree k=4. El entorno experimental se ha construido sobre Mininet, Open vSwitch, OpenFlow 1.3 y OpenDaylight, incorporando automatización en Python para desplegar topologías, instalar reglas de encaminamiento, ejecutar campañas de tráfico y generar evidencias reproducibles.

El desarrollo se ha planteado de forma incremental. En primer lugar, se valida el banco de pruebas mediante topologías de complejidad reducida. Posteriormente, se utiliza una arquitectura Leaf-Spine como etapa metodológica intermedia para estudiar mecanismos *multipath* bajo condiciones controladas. Finalmente, se implementa y evalúa una topología Fat-Tree k=4, sobre la que se ejecuta una campaña principal basada en *forwarding* determinista instalado desde OpenDaylight mediante RESTCONF y verificado posteriormente en el plano de datos mediante Open vSwitch.

La evaluación experimental incluye escenarios punto a punto, tráfico concurrente distribuido y tráfico concentrado hacia un único destino. Las métricas empleadas incluyen conectividad, latencia, *throughput*, contadores de flujos, grupos OpenFlow y buckets. Además, se desarrolla un demostrador ECMP limitado mediante grupos OpenFlow *select* y se documenta un intento exploratorio hacia ECMP completo en Fat-Tree. Los resultados muestran que la campaña RESTCONF determinista constituye una base estable y trazable, mientras que las extensiones *multipath* permiten validar mecanismos acotados e identificar límites técnicos en la activación completa de caminos en la jerarquía Fat-Tree.

El trabajo no pretende extrapolar prestaciones absolutas a una infraestructura física de producción, sino proporcionar una metodología reproducible para implementar, validar e interpretar arquitecturas SDN emuladas. Como resultado, se obtiene un banco de pruebas documentado, una campaña experimental defendible y una base sólida para futuras extensiones relacionadas con ECMP completo, telemetría, reconfiguración desatendida y escalabilidad hacia valores superiores de k.

Palabras clave: SDN, Fat-Tree, OpenDaylight, RESTCONF, Mininet, Open vSwitch, OpenFlow, ECMP.

ABSTRACT

This Bachelor's Thesis addresses the design, implementation, and evaluation of an SDN testbed for data centre network topologies, using a Fat-Tree k=4 topology as the main architecture. The experimental environment is built on Mininet, Open vSwitch, OpenFlow 1.3, and OpenDaylight, with Python-based automation to deploy topologies, install forwarding rules, run traffic campaigns, and generate reproducible evidence.

The work follows an incremental methodology. First, the testbed is validated through low-complexity topologies. Then, a Leaf-Spine architecture is used as an intermediate methodological stage to study multipath mechanisms under controlled conditions. Finally, a Fat-Tree k=4 topology is implemented and evaluated through a main campaign based on deterministic forwarding installed from OpenDaylight using RESTCONF and later verified in the data plane through Open vSwitch.

The experimental evaluation includes point-to-point scenarios, distributed concurrent traffic, and hotspot traffic towards a single destination. The metrics include connectivity, latency, throughput, flow counters, OpenFlow groups, and bucket counters. In addition, a limited ECMP demonstrator based on OpenFlow *select* groups is implemented, and an exploratory attempt towards full ECMP in Fat-Tree is documented. The results show that the deterministic RESTCONF campaign provides a stable and traceable baseline, while the multipath extensions validate bounded mechanisms and identify technical limitations in achieving complete path activation across the Fat-Tree hierarchy.

This work does not aim to extrapolate absolute performance values to a physical production infrastructure. Instead, it provides a reproducible methodology to implement, validate, and interpret emulated SDN architectures. As a result, the thesis delivers a documented testbed, a defensible experimental campaign, and a solid basis for future extensions related to full ECMP, telemetry, unattended reconfiguration, and scalability towards higher values of k.

Keywords: SDN, Fat-Tree, OpenDaylight, RESTCONF, Mininet, Open vSwitch, OpenFlow, ECMP.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

1. INTRODUCCIÓN Y MOTIVACIÓN

Las redes de comunicaciones actuales soportan una cantidad creciente de servicios, aplicaciones distribuidas y tráfico generado por usuarios, dispositivos y sistemas automatizados. En este contexto, las infraestructuras de centro de datos desempeñan un papel esencial, ya que concentran recursos de cómputo, almacenamiento y conectividad que deben operar con criterios de disponibilidad, escalabilidad y eficiencia. La topología de red utilizada en estos entornos condiciona directamente la forma en que los flujos atraviesan la infraestructura, la existencia de caminos alternativos y la capacidad de absorber patrones de tráfico concurrentes o concentrados [1], [2], [3].

Dentro de este ámbito, las arquitecturas Leaf-Spine y Fat-Tree se han consolidado como modelos representativos de topologías de redes de centro de datos. Ambas proporcionan redundancia de caminos y una estructura jerárquica o semijerárquica que facilita el análisis de estrategias de encaminamiento [1], [4], [5]. Sin embargo, esa riqueza topológica también introduce una mayor complejidad de control: no basta con disponer de enlaces alternativos, sino que es necesario definir cómo se instalan las reglas de *forwarding*, cómo se verifica que dichas reglas llegan al plano de datos y cómo se interpreta el tráfico que realmente circula por la red.

La aparición de SDN (*Software Defined Networking*) permite abordar este problema separando el plano de control del plano de datos. Bajo este paradigma, una controladora centralizada o lógicamente centralizada puede programar el comportamiento de los *switches* mediante protocolos como OpenFlow [6], [7], [8]. Esta separación resulta especialmente interesante en un trabajo experimental, porque permite construir un banco de pruebas donde la topología, las reglas de encaminamiento, las mediciones y las evidencias puedan automatizarse y documentarse de forma reproducible.

El presente TFG se enmarca en esa línea. El trabajo parte de una propuesta inicial orientada al estudio de topologías de centro de datos en entornos SDN, con especial atención a Fat-Tree, Leaf-Spine, automatización en Python y evaluación mediante métricas de red [9]. A partir de esa idea, el proyecto se ha concretado en la implementación de un banco de pruebas SDN basado en Mininet, Open vSwitch, OpenFlow 1.3 y OpenDaylight, tomando como arquitectura principal una topología Fat-Tree k=4 [10], [11], [12], [13]. La finalidad no es reproducir una infraestructura física de producción, sino construir un entorno emulado, controlado y trazable que permita estudiar de forma rigurosa la instalación de reglas, la conectividad y el comportamiento del tráfico bajo distintos escenarios.

1.1. Contexto y motivación

El crecimiento de servicios en la nube, aplicaciones distribuidas, plataformas de datos y sistemas conectados ha incrementado la importancia de las redes de centro de datos. En estas infraestructuras, el rendimiento no depende únicamente de la capacidad de los servidores, sino también de la forma en que los *hosts* se interconectan y de cómo se gestionan los caminos disponibles entre ellos. Una topología pobremente controlada puede provocar cuellos de botella, caminos infrautilizados o dificultades para diagnosticar el comportamiento real del tráfico [2], [3].

Las topologías tradicionales de centro de datos han evolucionado hacia diseños con mayor redundancia y menor dependencia de enlaces únicos. Leaf-Spine introduce una separación clara entre *switches* de acceso y *switches* de agregación superior, mientras que Fat-Tree organiza la red en *pods*, *switches* de borde, *switches* de agregación y *switches core* [1], [5]. En ambos casos, la disponibilidad de varios caminos entre pares de *hosts* abre la posibilidad de emplear estrategias *multipath*, pero también obliga a disponer de mecanismos de control y verificación más precisos.

SDN ofrece un marco adecuado para estudiar estas arquitecturas porque permite programar explícitamente el plano de datos. En lugar de depender únicamente del comportamiento distribuido de los dispositivos, la controladora puede instalar reglas OpenFlow y consultar información operacional [6], [7], [12]. No obstante, esta capacidad no elimina la necesidad de validación experimental. Una regla aceptada por la controladora no garantiza por sí sola que el tráfico atraviese el *datapath* previsto; por ello, resulta imprescindible contrastar el estado del controlador con evidencias obtenidas directamente de Open vSwitch, como *dumps* de flujos, contadores de paquetes, contadores de bytes y resultados de tráfico [11], [14].

La motivación principal de este trabajo es, por tanto, construir un banco de pruebas SDN que permita estudiar estas cuestiones de forma incremental. Antes de analizar una topología Fat-Tree, es necesario validar el entorno base, comprobar la conexión con la controladora, verificar topologías más simples y establecer un procedimiento de medición reproducible. Esta aproximación evita saltar directamente a una arquitectura compleja sin evidencias previas y permite que cada etapa del proyecto se apoye en resultados ya consolidados.

1.2. Planteamiento del problema

El problema abordado en este TFG no consiste únicamente en generar una topología Fat-Tree en Mininet. Ese sería solo el primer paso. El reto real es construir una arquitectura experimental completa en la que la topología pueda desplegarse, conectarse a una controladora SDN, programarse mediante reglas de *forwarding* y evaluarse con métricas de tráfico verificables.

En un entorno SDN emulado aparecen varias dificultades prácticas. En primer lugar, debe existir una correspondencia clara entre los elementos lógicos de la topología y los identificadores utilizados por el controlador. Los nombres de los *switches* en Mininet y Open vSwitch no tienen por qué coincidir directamente con los identificadores `openflow:<id>` utilizados por OpenDaylight, por lo que es necesario documentar y verificar ese mapeo [10], [11], [15].

En segundo lugar, la instalación de reglas debe ser trazable. Si las reglas se instalan manualmente en OVS, se obtiene una validación del *datapath*, pero no una validación completa del flujo controlador-*datapath*. Por ese motivo, la campaña principal del trabajo se apoya en OpenDaylight RESTCONF como mecanismo de instalación, manteniendo siempre una verificación posterior en OVS [14], [15], [16].

En tercer lugar, la interpretación de mecanismos *multipath* exige especial cuidado. La presencia de caminos alternativos en una topología Fat-Tree no implica automáticamente que exista balanceo de carga. Del mismo modo, la instalación de grupos OpenFlow *select* no basta para afirmar ECMP completo. Es necesario comprobar si los *buckets* de dichos grupos reciben tráfico y en qué nivel de la jerarquía se produce realmente el reparto [12], [17], [18], [19].

Por último, el trabajo debe mantener un equilibrio entre ambición técnica y defendibilidad académica. Algunas extensiones, como Containernet, modificaciones topológicas tipo *subways*/*superways* o reconfiguración automática desatendida, son líneas de evolución naturales, pero incorporarlas sin una validación completa podría debilitar la coherencia del proyecto. Por ello, el TFG delimita un alcance experimental concreto y reserva esas ampliaciones para trabajo futuro.

1.3. Objetivos del trabajo

El objetivo general del TFG es diseñar, implementar y evaluar un banco de pruebas SDN para topologías de centro de datos, tomando como arquitectura principal una topología Fat-Tree k=4, con validación incremental, automatización en Python y análisis de métricas de tráfico en un entorno emulado reproducible.

A partir de este objetivo general, se definen los siguientes objetivos específicos:

1. Construir un banco de pruebas SDN basado en Linux, Mininet, Open vSwitch, OpenFlow y OpenDaylight.

2. Validar progresivamente el entorno mediante topologías de complejidad creciente antes de abordar Fat-Tree.

3. Diseñar e implementar una topología Fat-Tree parametrizable, empleando k=4 como instancia experimental manejable.

4. Instalar políticas de *forwarding* desde la controladora SDN y verificar su efecto real en el plano de datos.

5. Evaluar la topología Fat-Tree con métricas de latencia, rendimiento y evidencias de tráfico reproducibles.

6. Explorar mecanismos *multipath* mediante grupos OpenFlow *select*, delimitando con precisión su alcance experimental.

7. Documentar las limitaciones del banco de pruebas y formular líneas futuras coherentes con la propuesta inicial.

La Tabla 1.1 resume la relación entre estos objetivos y los capítulos de la memoria. Su finalidad es proporcionar una trazabilidad inicial del documento, no evaluar todavía el grado final de cumplimiento. Esa evaluación se realizará en el capítulo de conclusiones, una vez presentados los resultados y la discusión.

Tabla 1.1: Trazabilidad inicial entre los objetivos específicos del TFG y los capítulos de la memoria

Obj.	**Objetivo específico**	**Tratamiento en la memoria**	**Evidencia principal**
OE1	Construir un banco de pruebas SDN basado en Linux, Mininet, Open vSwitch, OpenFlow y OpenDaylight.	Capítulos 3 y 4	Arquitectura del entorno, configuración de OVS, conexión con la controladora y validación incremental inicial.
OE2	Validar progresivamente el entorno mediante topologías de complejidad creciente antes de abordar Fat-Tree.	Capítulos 4 y 5	Topologías básica e intermedia, y etapa Leaf-Spine como puente metodológico hacia arquitecturas de centro de datos.
OE3	Diseñar e implementar una topología Fat-Tree parametrizable, utilizando k=4 como instancia experimental manejable.	Capítulo 6	Generador Fat-Tree, nomenclatura de *hosts* y *switches*, estructura por *pods* y análisis teórico de escalabilidad.
OE4	Instalar políticas de *forwarding* desde la controladora SDN y verificar su efecto real en el plano de datos.	Capítulos 3, 7 y 8	Uso de OpenDaylight RESTCONF, reglas OpenFlow, verificaciones en OVS y contadores de flujos.
OE5	Evaluar la topología Fat-Tree con métricas de latencia, rendimiento y evidencias de tráfico reproducibles.	Capítulos 7 y 8	Campañas punto a punto, tráfico concurrente, escenario *hotspot*, RTT, *throughput* y contadores del *datapath*.
OE6	Explorar mecanismos *multipath* mediante grupos OpenFlow *select*, delimitando con precisión su alcance experimental.	Capítulos 8 y 9	Demostrador ECMP limitado, migración RESTCONF e intento exploratorio hacia ECMP completo sin sobreinterpretar los resultados.
OE7	Documentar las limitaciones del banco de pruebas y formular líneas futuras coherentes con la propuesta inicial.	Capítulos 9 y 10	Discusión crítica, amenazas a la validez, matriz de cumplimiento frente a la propuesta y trabajo futuro.

Nota: La tabla ofrece una trazabilidad inicial de la memoria. La evaluación final del grado de cumplimiento respecto a la propuesta original se reserva para el Capítulo 10.

1.4. Alcance y delimitaciones metodológicas

El alcance experimental del trabajo se centra en una instancia Fat-Tree k=4 desplegada en Mininet y controlada mediante OpenDaylight. Esta decisión permite trabajar con una topología suficientemente rica para representar *pods*, *switches* de borde, *switches* de agregación, *switches core* y caminos inter-*pod*, pero sin introducir una carga excesiva para el entorno local de emulación [1], [10].

La propuesta inicial contemplaba un marco amplio de posibilidades, incluyendo el estudio de topologías de centro de datos, automatización en Python, exploración de la dimensión del parámetro k, análisis de métricas y posibles extensiones relacionadas con Containernet, variantes topológicas y reconfiguración desatendida [9]. Durante el desarrollo del TFG, ese marco se concretó en una línea principal más acotada: implementación SDN reproducible de Fat-Tree, validación incremental, campaña RESTCONF determinista y exploración *multipath* controlada.

Esta delimitación responde a una decisión metodológica. En un TFG de carácter experimental, resulta preferible consolidar una campaña trazable y defendible antes que incorporar muchas extensiones parcialmente validadas. Por ello, el trabajo distingue entre resultados principales, extensiones acotadas y líneas futuras. La campaña RESTCONF determinista constituye la evidencia principal; el ECMP limitado se presenta como demostrador *multipath* validado en un escenario concreto; y el intento hacia ECMP completo se documenta como exploración parcial, sin afirmarlo como una solución final.

Estas delimitaciones no deben interpretarse como una reducción arbitraria del trabajo, sino como una forma de preservar su rigor. Mininet clásico se ajusta mejor al estudio de una arquitectura cableada de centro de datos que Mininet-wifi, y la combinación con OVS y OpenFlow permite observar reglas, puertos y contadores en el plano de datos [10], [11], [12]. La integración de Containernet o Docker habría añadido una capa adicional de dependencias sin mejorar necesariamente la evidencia principal sobre Fat-Tree y SDN. Las variantes *subway*/*superway* y la reconfiguración automática desatendida requerirían una nueva fase de diseño, telemetría, decisión y reprogramación dinámica que excede el cierre experimental del proyecto.

La Tabla 1.2 recoge las principales delimitaciones adoptadas y su impacto en la memoria.

Tabla 1.2: Alcance y delimitaciones metodológicas adoptadas en el desarrollo del TFG

Elemento	**Tratamiento adoptado**	**Justificación**	**Impacto en la memoria**
Entorno de emulación	Uso de Mininet clásico con Open vSwitch y OpenFlow 1.3.	El foco final del trabajo es una arquitectura cableada de centro de datos, no un escenario inalámbrico.	Permite mantener un banco de pruebas más estable, reproducible y coherente con Fat-Tree y Leaf-Spine.
Controladora SDN	Uso de OpenDaylight con RESTCONF para la instalación y verificación de reglas.	RESTCONF aporta una interfaz programable y trazable entre automatización, controlador y *datapath*.	La campaña principal del Capítulo 8 se apoya en reglas instaladas desde la controladora.
Dimensión de Fat-Tree	Evaluación experimental sobre k=4, complementada con análisis estructural para valores superiores.	k=4 ofrece una instancia completa, pero manejable dentro de un entorno emulado local.	El Capítulo 6 recoge la escalabilidad teórica y el Capítulo 8 evalúa la instancia experimental.
Containernet y Docker	No se integran en la campaña experimental final.	Su incorporación habría añadido dependencias y riesgo operativo sin mejorar proporcionalmente la evidencia principal.	Se plantean como una extensión futura para escenarios con servicios contenedorizados.
Subways y *superways*	No se implementan como modificaciones topológicas reales.	Requerirían rediseñar la topología, reprogramar rutas y repetir campañas ya consolidadas.	La exploración *multipath* se aborda mediante grupos OpenFlow *select* y queda delimitada experimentalmente.
Reconfiguración desatendida	No se implementa un algoritmo completo de detección y actuación automática.	Exige una capa adicional de telemetría, decisión y reprogramación dinámica que excede el cierre experimental del TFG.	Se formula como línea futura apoyada en las evidencias de contadores y tráfico obtenidas.
ECMP completo	Se realiza un intento exploratorio controlado, no una implementación completa de producción.	La activación *multipath* observada no fue suficiente para afirmar reparto completo en toda la jerarquía Fat-Tree.	El Capítulo 9 lo interpreta como resultado parcial y el Capítulo 10 lo recogerá como trabajo futuro.
Métricas experimentales	Uso de RTT, *throughput*, conectividad, contadores de flujos, grupos y *buckets*.	Estas métricas permiten valorar funcionamiento, estabilidad y uso efectivo del *datapath*.	Los resultados se interpretan como validación funcional en emulación, no como rendimiento extrapolable a hardware real.

1.5. Contribuciones principales

Las contribuciones principales del TFG pueden resumirse en los siguientes puntos:

- Diseño de un banco de pruebas SDN reproducible basado en Mininet, Open vSwitch, OpenFlow 1.3 y OpenDaylight.
- Validación incremental del entorno mediante topologías de complejidad creciente antes de abordar la arquitectura Fat-Tree.
- Implementación parametrizada en Python de una topología Fat-Tree, con nomenclatura, direccionamiento, política de puertos y metadatos trazables.
- Ejecución de una campaña experimental principal basada en *forwarding* determinista instalado desde OpenDaylight RESTCONF.

- Evaluación de escenarios de tráfico punto a punto, tráfico concurrente distribuido y tráfico concentrado hacia un único destino.
- Generación automatizada de artefactos, tablas, figuras y manifiestos que permiten reconstruir la relación entre campaña, datos brutos, datos procesados y resultados integrados en la memoria.
- Implementación de un demostrador ECMP limitado mediante grupos OpenFlow *select*, incluyendo una variante instalada mediante RESTCONF.
- Realización de un intento exploratorio hacia ECMP completo en Fat-Tree k=4, documentando tanto la viabilidad parcial como las limitaciones observadas.

Estas contribuciones se formulan de manera deliberadamente prudente. El proyecto no pretende demostrar prestaciones absolutas de una infraestructura física ni presentar un sistema autónomo de optimización SDN. Su aportación se centra en la construcción de un banco de pruebas controlado, la validación rigurosa de una arquitectura Fat-Tree y la interpretación honesta de las extensiones *multipath* evaluadas.

1.6. Organización de la memoria

La memoria se estructura de forma incremental, siguiendo la misma lógica empleada durante el desarrollo experimental.

El Capítulo 2 presenta los fundamentos teóricos y el estado del arte relacionado con SDN, OpenFlow, controladores, Mininet, Open vSwitch, topologías de centro de datos, Fat-Tree, Leaf-Spine y mecanismos *multipath*. Este capítulo proporciona el contexto técnico necesario para interpretar las decisiones de diseño posteriores.

El Capítulo 3 describe el banco de pruebas y la arquitectura de la solución. En él se detallan los componentes empleados, la relación entre Mininet, OVS y OpenDaylight, la estructura del proyecto y los mecanismos de automatización utilizados para preservar la trazabilidad.

El Capítulo 4 recoge la validación inicial del entorno mediante topologías más simples. Su objetivo es demostrar que el banco de pruebas funciona antes de introducir arquitecturas de mayor complejidad.

El Capítulo 5 presenta la etapa Leaf-Spine como puente metodológico hacia Fat-Tree. Esta fase permite estudiar de forma controlada la existencia de caminos alternativos y la lógica *multipath* en una arquitectura más sencilla.

El Capítulo 6 desarrolla el diseño e implementación de la topología Fat-Tree. Incluye el modelo estructural, la instancia k=4, la implementación parametrizada en Python, los metadatos, la política de puertos, la representación visual y el análisis de escalabilidad teórica.

El Capítulo 7 define la metodología experimental. En él se concretan los escenarios de tráfico, las métricas, el procedimiento de ejecución, el tratamiento de resultados y las limitaciones de validez que deben tenerse en cuenta.

El Capítulo 8 presenta los resultados experimentales obtenidos sobre la topología Fat-Tree k=4. Incluye la campaña RESTCONF determinista, el demostrador ECMP limitado y el intento exploratorio hacia ECMP completo.

El Capítulo 9 discute críticamente los resultados, comparando las estrategias de *forwarding*, interpretando las evidencias *multipath* y delimitando el alcance real de las conclusiones.

Por último, el Capítulo 10 recoge las conclusiones finales, evalúa el grado de cumplimiento de los objetivos y plantea las líneas de trabajo futuro, incluyendo aquellas premisas de la propuesta inicial que no se han implementado experimentalmente en esta versión del proyecto.

Con esta organización, la memoria mantiene una secuencia lógica: primero se justifica el problema, después se construye y valida el banco de pruebas, posteriormente se implementa y evalúa Fat-Tree, y finalmente se discuten las evidencias obtenidas antes de formular las conclusiones.

2. FUNDAMENTOS Y ESTADO DEL ARTE

Este capítulo presenta los fundamentos técnicos y el estado del arte que sustentan el desarrollo del proyecto. Su objetivo no es realizar una revisión bibliográfica exhaustiva de todas las líneas existentes en redes definidas por software, sino establecer el marco conceptual necesario para entender las decisiones de diseño adoptadas en los capítulos posteriores. En particular, se revisan los principios de SDN, el papel de OpenFlow como interfaz de control del plano de datos, la función de los controladores SDN y de RESTCONF, el uso de Mininet y Open vSwitch como entorno experimental, las topologías de centro de datos Leaf-Spine y Fat-Tree, y los mecanismos de encaminamiento *multipath* basados en ECMP y grupos OpenFlow *select* [1], [6], [7], [8], [10], [11], [12], [16], [17].

La organización del capítulo sigue una progresión deliberada. En primer lugar, se introduce el paradigma SDN como separación entre plano de control y plano de datos. A continuación, se describen las tecnologías concretas utilizadas en el banco de pruebas del proyecto. Posteriormente, se contextualizan las topologías de centro de datos que motivan la elección de Leaf-Spine como etapa metodológica intermedia y Fat-Tree como arquitectura principal. Finalmente, se revisa el concepto de *multipath* y su implementación mediante grupos OpenFlow *select*, que constituye la base conceptual para interpretar los resultados experimentales del Capítulo 8.

2.1. Redes definidas por software

Las redes definidas por software, o *Software Defined Networking* (SDN), surgen como respuesta a la rigidez de las arquitecturas de red tradicionales. En una red convencional, cada dispositivo de encaminamiento o conmutación integra tanto la lógica de decisión como la ejecución del reenvío de paquetes. Esta integración dificulta la programación global de la red, la automatización de políticas y la introducción de mecanismos avanzados de control del tráfico [7], [8].

El principio fundamental de SDN consiste en desacoplar el plano de control del plano de datos. El plano de control se encarga de tomar decisiones de encaminamiento, instalar políticas y mantener una visión lógica del estado de la red, mientras que el plano de datos ejecuta las reglas de reenvío sobre los paquetes que atraviesan los dispositivos. Esta separación permite que el comportamiento de la red sea programable desde una entidad lógica centralizada, denominada controladora SDN, sin que ello implique necesariamente una centralización física absoluta [6], [7], [8].

La separación entre plano de aplicación, plano de control y plano de datos es el eje conceptual que permite interpretar la arquitectura empleada en este TFG. En el proyecto,

el plano de aplicación se materializa en los *scripts* de automatización y procesado; el plano de control se representa mediante OpenDaylight; y el plano de datos se implementa mediante *switches* Open vSwitch desplegados en Mininet. Esta correspondencia entre capas conceptuales y componentes experimentales se ilustra en la Figura 2.1.

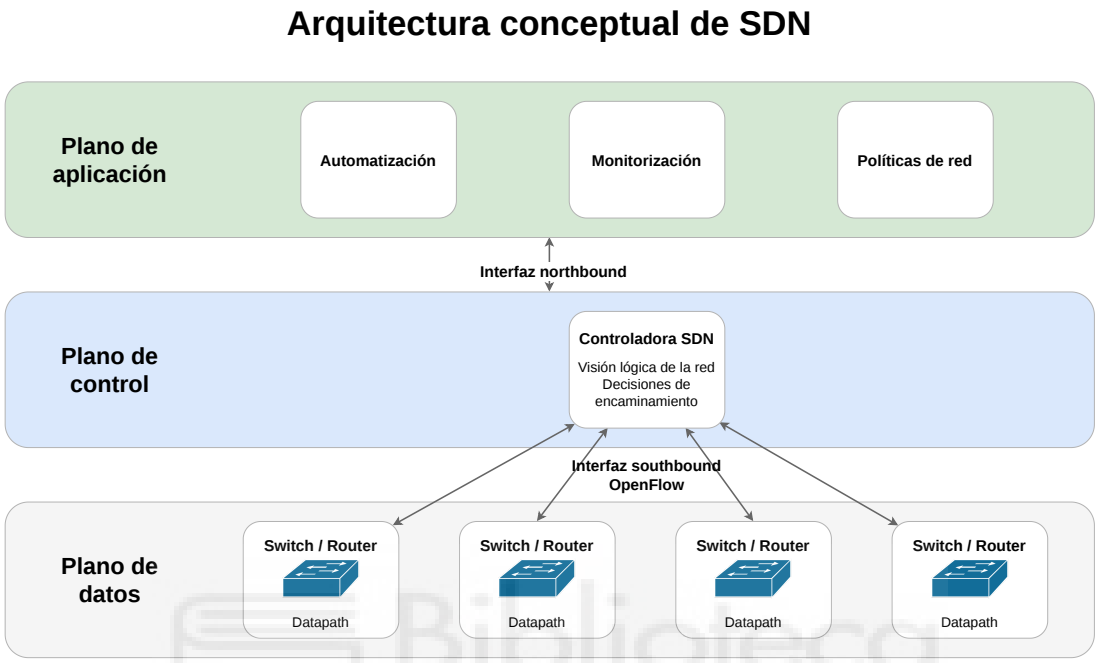

Figura 2.1: Arquitectura conceptual de SDN y separación entre plano de aplicación, plano de control y plano de datos.

Esta arquitectura resulta especialmente adecuada para entornos en los que se requiere automatización, trazabilidad y capacidad de experimentación. En el contexto de este TFG, SDN permite instalar reglas explícitas de encaminamiento, consultar el estado de los dispositivos y validar experimentalmente topologías con caminos múltiples. Por tanto, el interés de SDN en este trabajo no se limita a su valor teórico, sino que se materializa en una metodología experimental basada en reglas de *forwarding* reproducibles y verificables.

2.2. OpenFlow como interfaz hacia el plano de datos

OpenFlow es uno de los protocolos más representativos del paradigma SDN. Su función principal es permitir que una controladora programe el comportamiento de los dispositivos del plano de datos mediante la instalación, modificación y eliminación de reglas de flujo. Cada regla puede especificar condiciones de coincidencia sobre los paquetes, prioridades, contadores y acciones asociadas, como reenviar por un puerto, descartar tráfico o invocar un grupo OpenFlow [6], [12].

Desde el punto de vista de este proyecto, OpenFlow es relevante por tres motivos. En primer lugar, permite sustituir el aprendizaje tradicional de nivel 2 por reglas explícitas de

forwarding, lo que facilita la reproducibilidad experimental. En segundo lugar, ofrece contadores de paquetes y bytes que permiten verificar si una regla ha sido utilizada realmente durante una prueba. En tercer lugar, incorpora mecanismos como los grupos OpenFlow *select*, que permiten representar alternativas de salida para aproximar comportamientos *multipath* [12].

La versión utilizada en el banco de pruebas es OpenFlow 1.3, soportada por Open vSwitch y por el controlador OpenDaylight empleado en el proyecto. Esta elección resulta importante porque los grupos OpenFlow y sus tipos asociados, entre ellos *select*, forman parte del mecanismo con el que se han construido los demostradores ECMP limitados y los intentos exploratorios posteriores. No obstante, debe destacarse que la existencia de soporte OpenFlow no implica automáticamente que cualquier política de encaminamiento avanzado pueda implementarse de forma completa o transparente. La validez experimental depende de la instalación correcta de reglas, de su aparición en el *datapath*, de la conectividad resultante y de la interpretación de los contadores observados.

2.3. Controladores SDN, OpenDaylight y RESTCONF

Una controladora SDN proporciona la lógica de control encargada de gestionar el comportamiento de los dispositivos del plano de datos. En el proyecto se utiliza OpenDaylight como controladora, por tratarse de una plataforma modular con soporte para OpenFlow y exposición de interfaces de gestión mediante RESTCONF [13], [15]. Esta combinación permite que el banco de pruebas no dependa únicamente de comandos locales sobre los *switches* virtuales, sino que pueda instalar y consultar información a través de una controladora SDN real.

RESTCONF proporciona una interfaz HTTP para acceder y modificar datos estructurados del controlador de acuerdo con modelos YANG [16]. En este TFG, RESTCONF se utiliza como mecanismo de interacción entre los *scripts* de automatización y OpenDaylight. Esta decisión tiene una implicación metodológica relevante: las reglas de encaminamiento no se instalan únicamente desde el propio *datapath*, sino que se envían desde una capa de control programable, quedando registradas y verificadas tanto en la controladora como en Open vSwitch.

OpenDaylight también permite consultar inventarios operacionales, nodos OpenFlow descubiertos y, cuando procede, información asociada a flujos y grupos instalados [15]. En consecuencia, el controlador no se emplea como un elemento decorativo del banco de pruebas, sino como parte activa de la trazabilidad experimental. En los capítulos posteriores, esta trazabilidad se materializa en capturas de inventario, validaciones de conectividad, instalación de flujos mediante RESTCONF y comprobación de contadores en el plano de datos.

2.4. Mininet y Open vSwitch como entorno de emulación

Mininet es una herramienta ampliamente utilizada para emular redes definidas por software en un único sistema físico. Permite crear *hosts*, *switches*, enlaces y controladores de forma programable, facilitando la experimentación con topologías complejas sin necesidad de desplegar hardware dedicado [10], [20]. En este proyecto, Mininet se utiliza para instanciar topologías de complejidad creciente, desde escenarios básicos hasta una topología Fat-Tree *k*-aria.

Open vSwitch (OVS) actúa como *switch* virtual programable compatible con OpenFlow. Su papel es esencial porque constituye el *datapath* sobre el que se instalan las reglas de *forwarding*, se ejecutan las acciones de salida y se obtienen contadores de paquetes y bytes [11], [14]. La combinación Mininet–OVS permite reproducir un entorno SDN completo: Mininet crea la topología y los *hosts* emulados, OVS ejecuta las reglas en los *switches* virtuales, OpenDaylight actúa como controladora y los *scripts* Python automatizan la instalación, validación y procesado de resultados.

La arquitectura tecnológica del banco de pruebas se apoya en la interacción entre estos componentes. La automatización Python interactúa con OpenDaylight mediante RESTCONF/HTTP, OpenDaylight comunica con los *switches* OVS mediante OpenFlow 1.3, Mininet proporciona el entorno de emulación y las evidencias experimentales se obtienen a partir de contadores, *dumps*, pruebas de conectividad, tráfico `iperf3`, ficheros CSV y figuras generadas automáticamente. La relación entre estas piezas se representa de forma esquemática en la Figura 2.2.

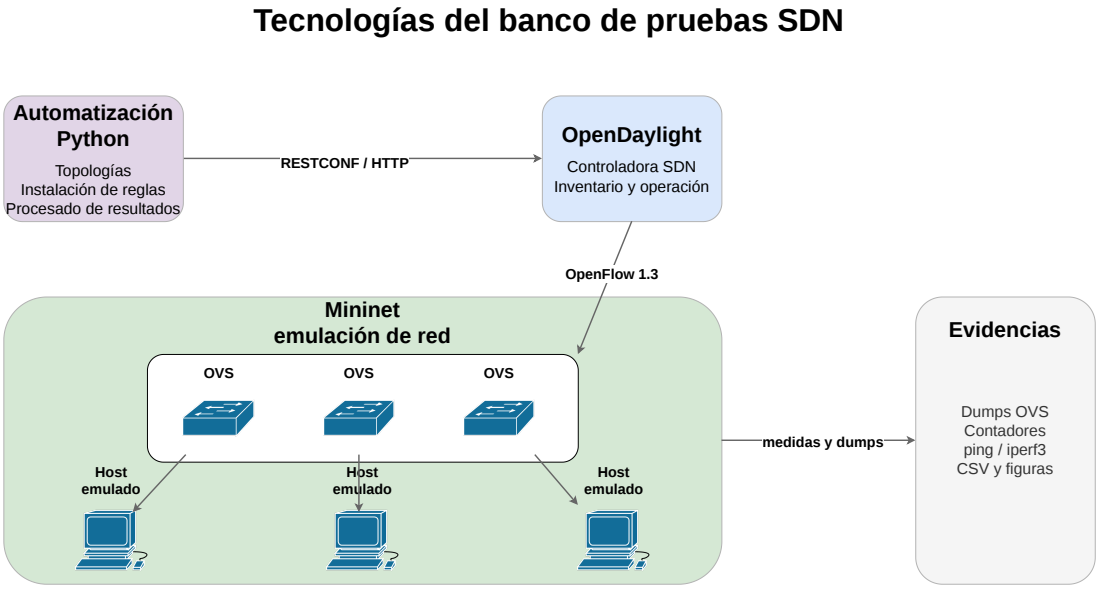

Figura 2.2: Relación conceptual entre las tecnologías utilizadas en el banco de pruebas SDN.

Esta relación tecnológica justifica el enfoque adoptado en el proyecto. Frente a una validación basada únicamente en conectividad básica, el banco de pruebas se orienta a

obtener evidencias reproducibles: reglas instaladas, nodos descubiertos, contadores no nulos, resultados de tráfico y artefactos procesados. De este modo, la emulación no se interpreta como una simple demostración visual de topologías, sino como un procedimiento experimental trazable.

2.5. Topologías de centro de datos

Las topologías de centro de datos se diseñan para proporcionar conectividad escalable, redundancia y capacidad de comunicación entre un gran número de servidores. En este contexto, las arquitecturas jerárquicas tradicionales pueden presentar cuellos de botella si no se dimensionan adecuadamente, especialmente cuando el tráfico este-oeste entre servidores adquiere mayor relevancia que el tráfico norte-sur hacia el exterior. Por ello, las arquitecturas modernas tienden a explotar múltiples caminos entre pares de *hosts* y a distribuir el tráfico a través de enlaces equivalentes [1], [2], [3].

Dentro de este marco, Leaf-Spine y Fat-Tree representan dos diseños especialmente relevantes. Leaf-Spine se basa en dos capas: una capa *leaf*, a la que se conectan los *hosts*, y una capa *spine*, a la que se conectan los *switches leaf*. En una arquitectura Leaf-Spine canónica, cada *switch leaf* se conecta con todos los *switches spine*, proporcionando varios caminos posibles entre hojas distintas [4], [5]. Esta estructura resulta conceptualmente sencilla y permite introducir mecanismos *multipath* sin la complejidad jerárquica completa de una Fat-Tree.

Fat-Tree, por su parte, es una topología jerárquica *k*-aria organizada en *pods*, *switches* de borde, *switches* de agregación y *switches core*. La propuesta de Fat-Tree para redes de centro de datos se asocia a la posibilidad de construir arquitecturas escalables utilizando elementos de conmutación básicos y múltiples caminos entre *hosts* [1]. En este TFG, Fat-Tree constituye la arquitectura principal de evaluación, mientras que Leaf-Spine se utiliza como etapa metodológica intermedia para validar previamente la lógica *multipath* en un escenario más reducido.

La comparación entre Leaf-Spine y Fat-Tree permite entender la progresión metodológica del proyecto. La primera arquitectura proporciona un escenario de dos capas útil para comprobar conectividad, caminos alternativos y reglas de *forwarding* en una escala reducida. La segunda introduce una estructura jerárquica más rica, organizada por *pods*, donde la interpretación de caminos, reglas y contadores resulta más exigente. Esta diferencia estructural se ilustra en la Figura 2.3.

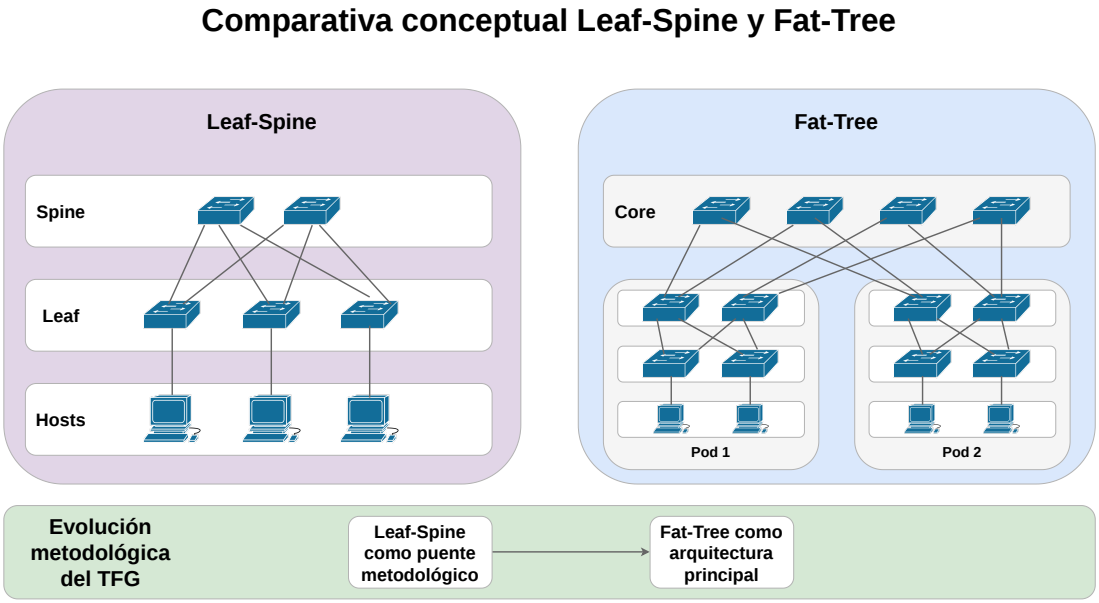

Figura 2.3: Comparativa conceptual entre una arquitectura Leaf-Spine y una topología Fat-Tree k-aria.

Desde el punto de vista experimental, esta progresión evita abordar directamente el caso más complejo sin haber validado antes los elementos básicos del banco de pruebas. La topología Leaf-Spine permite comprobar conectividad, instalación de reglas, utilización de enlaces alternativos y comportamiento de grupos OpenFlow en una escala reducida. Fat-Tree, en cambio, permite estudiar cómo esos mismos principios se comportan en una arquitectura más jerárquica, con más *switches*, más niveles y mayor complejidad de interpretación.

2.6. Fat-Tree k-aria y escalabilidad estructural

Una Fat-Tree k-aria se caracteriza por una estructura regular organizada en k *pods*. Cada *pod* contiene *switches* de borde y agregación, mientras que la capa *core* conecta los *pods* entre sí, de modo que todos los *switches* de *core* se conectan a algún *switch* de agregación dentro de cada *pod*. Para un valor par de k, el número de *hosts*, *switches* y enlaces crece de forma estructurada, lo que permite analizar la escalabilidad teórica de la topología [1], [4]. En este proyecto se emplea k=4 como instancia experimental manejable dentro de un entorno emulado, con 16 *hosts* y 20 *switches*.

La elección de k=4 responde a un compromiso metodológico. Por un lado, la topología es suficientemente rica para incluir múltiples *pods*, *switches core*, enlaces inter-*pod* y caminos alternativos. Por otro lado, mantiene un tamaño compatible con la ejecución en un equipo local, la captura de evidencias y la interpretación manual de los resultados. Valores superiores del parámetro k serían útiles para un análisis de escalabilidad más amplio,

pero aumentarían significativamente el número de nodos, enlaces, reglas y contadores que habría que instalar y verificar.

En consecuencia, el proyecto no plantea Fat-Tree k=4 como una red de producción ni como un entorno equivalente a un centro de datos real, sino como una instancia experimental defendible. Su función es permitir el estudio reproducible de mecanismos SDN sobre una arquitectura jerárquica con *multipath*, manteniendo el control sobre las reglas instaladas y sobre las evidencias generadas.

2.7. Multipath, ECMP y grupos OpenFlow select

El uso de múltiples caminos entre origen y destino es una característica habitual en topologías de centro de datos. Cuando existen rutas de coste equivalente, ECMP permite distribuir tráfico entre ellas con el objetivo de aprovechar mejor la capacidad agregada de la red. Sin embargo, ECMP no debe interpretarse como sinónimo de balanceo perfecto. El reparto efectivo depende de la granularidad de los flujos, de los campos utilizados por el algoritmo de selección, de la implementación del *datapath* y de la diversidad real del tráfico observado [17], [19].

En OpenFlow, una forma de aproximar este comportamiento consiste en utilizar grupos de tipo *select*. Una regla de flujo puede invocar un grupo, y dicho grupo contiene varios *buckets*. Cada *bucket* incluye una o varias acciones, típicamente una acción de salida por un puerto concreto [12], [18]. Por tanto, es importante distinguir entre tres conceptos: el grupo representa el conjunto de alternativas, el *bucket* representa una alternativa de acción dentro de ese grupo y el puerto es la interfaz concreta por la que se reenvía el paquete. Esta distinción resulta esencial para interpretar los resultados experimentales del proyecto.

El mecanismo empleado en el proyecto se apoya en esta idea: un *switch* de entrada recibe tráfico desde un *host* origen y utiliza un grupo *select* con varios *buckets*. Cada *bucket* define una salida asociada a un camino alternativo. Los caminos atraviesan *switches* intermedios y convergen en un *switch* de salida antes de llegar al *host* destino. La Figura 2.4 representa esta lógica de forma conceptual.

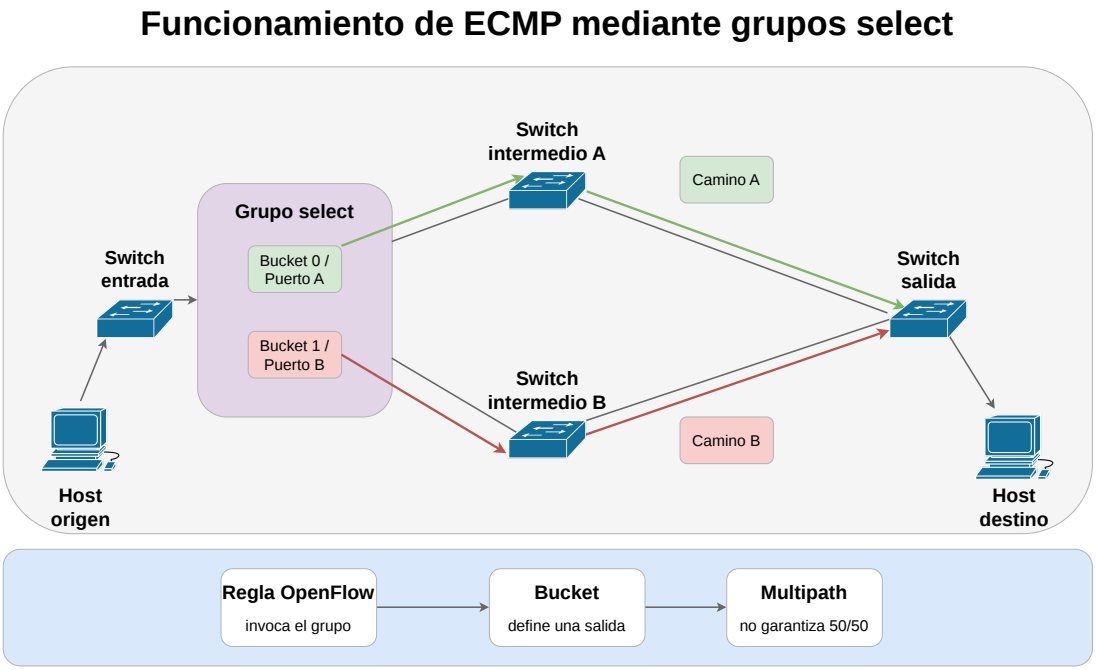

Figura 2.4: Representación conceptual del encaminamiento *multipath* mediante un grupo OpenFlow *select*.

Esta precisión conceptual es relevante para la interpretación de este TFG. En los experimentos posteriores, la activación de *buckets* y el uso de ramas alternativas se consideran evidencias de *multipath*. Sin embargo, no se identifican automáticamente con ECMP completo en toda la topología ni con balanceo dinámico perfecto. Esta prudencia metodológica es especialmente importante en Fat-Tree, donde la existencia de caminos alternativos no garantiza por sí sola que todos los niveles jerárquicos activen de forma equilibrada sus *buckets* en una campaña concreta.

2.8. Relación del estado del arte con el enfoque del proyecto

La literatura sobre SDN, OpenFlow, controladores programables, emulación de redes y topologías de centro de datos proporciona la base técnica sobre la que se construye este proyecto [1], [6], [7], [10], [11]. No obstante, el alcance del TFG se sitúa en un punto concreto: no se pretende diseñar un nuevo controlador SDN, proponer un algoritmo de balanceo original ni demostrar una arquitectura de producción a gran escala. El objetivo es implementar y validar de forma reproducible un banco de pruebas SDN sobre topologías jerárquicas, con especial atención a Fat-Tree k=4 y a la trazabilidad de las evidencias.

Desde esta perspectiva, los capítulos posteriores trasladan los fundamentos revisados a una implementación concreta. El Capítulo 3 describe el banco de pruebas y la arquitectura de la solución. El Capítulo 4 valida progresivamente el entorno experimental. El Capítulo 5 utiliza Leaf-Spine como etapa metodológica intermedia para estudiar *multipath* en un

escenario reducido. El Capítulo 6 formaliza el diseño de la topología Fat-Tree. El Capítulo 7 define la metodología de pruebas, y el Capítulo 8 presenta los resultados obtenidos.

El marco teórico desarrollado en este capítulo permite justificar tres decisiones centrales del proyecto. En primer lugar, la utilización de SDN y OpenFlow se justifica por la necesidad de programar reglas explícitas y verificables. En segundo lugar, el uso de OpenDaylight, RESTCONF, Mininet y Open vSwitch se justifica por su adecuación para construir un banco de pruebas emulado, automatizable y trazable. En tercer lugar, la elección de Leaf-Spine y Fat-Tree se justifica por su relación con arquitecturas de centro de datos y por su capacidad para introducir escenarios con múltiples caminos. Esta línea conecta, además, con la propuesta inicial del TFG y con antecedentes académicos próximos sobre arquitecturas Leaf-Spine y Fat-Tree en el entorno de la UMH [4], [9], [21].

2.9. Síntesis del capítulo

En este capítulo se han presentado los conceptos necesarios para interpretar el desarrollo experimental del TFG. SDN proporciona la separación entre plano de control y plano de datos. OpenFlow permite programar reglas de *forwarding* sobre *switches* compatibles. OpenDaylight y RESTCONF ofrecen una vía de interacción entre automatización y controladora. Mininet y Open vSwitch permiten emular topologías y ejecutar reglas sobre un *datapath* verificable. Leaf-Spine y Fat-Tree aportan el contexto topológico de centro de datos. Finalmente, ECMP y los grupos OpenFlow *select* permiten introducir el problema de los caminos múltiples y la necesidad de interpretar con cautela la activación de *buckets* y el reparto de tráfico.

Esta base teórica establece el puente entre el estado del arte y la implementación propia del proyecto. A partir de este punto, el documento deja de centrarse en los conceptos generales y pasa a describir el banco de pruebas desarrollado, sus componentes, su estructura de automatización y los criterios utilizados para generar evidencias reproducibles.

3. BANCO DE PRUEBAS Y ARQUITECTURA

3.1. Entorno experimental

El banco de pruebas desarrollado en este trabajo se construyó sobre un entorno de emulación SDN orientado a la reproducibilidad, al control explícito del plano de datos y a la obtención de evidencias verificables. La elección del entorno no tuvo como objetivo reproducir un centro de datos real a escala industrial, sino disponer de una plataforma suficientemente controlada para desplegar topologías, instalar políticas de *forwarding*, generar tráfico de validación y conservar trazas técnicas del comportamiento observado.

La solución se apoya en una combinación de herramientas ampliamente utilizadas en entornos académicos de redes definidas por software: Mininet para la emulación de topologías, Open vSwitch como *datapath* OpenFlow, OpenDaylight como controladora SDN, y una capa de automatización propia desarrollada en Python para coordinar despliegue, validación, empuje de reglas y generación de evidencias.

3.1.1. Sistema operativo y requisitos de hardware

El entorno experimental se ejecutó sobre Ubuntu 24.04.4 LTS. Esta elección permitió trabajar con versiones actuales del sistema operativo, mantener compatibilidad con Mininet y Open vSwitch, y disponer de un entorno GNU/Linux adecuado para ejecutar OpenDaylight, scripts de automatización y herramientas de diagnóstico del sistema.

La reproducibilidad del entorno requiere identificar con precisión las versiones principales del sistema operativo, de las herramientas SDN y de las dependencias utilizadas durante la construcción y validación del banco de pruebas. Esta información se recoge en la Tabla 3.1.

Tabla 3.1: Versiones principales del entorno experimental utilizado en el banco de pruebas

Componente	**Versión**	**Observación**
Sistema operativo	Ubuntu 24.04.4 LTS	Entorno base de experimentación sobre GNU/Linux.
Java	OpenJDK 21.0.10	Máquina virtual Java utilizada para la ejecución de OpenDaylight.
Mininet	2.3.0	Plataforma de emulación de topologías de red.
Open vSwitch	3.3.4	Implementación software del *datapath* OpenFlow.
OpenDaylight	Titanium 0.22.0	Controladora SDN utilizada en el proyecto.
OpenFlow	1.3	Protocolo validado y utilizado para la comunicación con los *switches* Open vSwitch.

Desde el punto de vista metodológico, el requisito principal no fue maximizar la capacidad de cómputo, sino asegurar que el entorno fuese estable, inspeccionable y suficientemente reproducible. Por ello, además de registrar las versiones principales, se incorporaron *snapshots* del entorno mediante la utilidad `doctor`, como se describe más adelante en este capítulo.

3.1.2. OpenDaylight, Open vSwitch y Mininet

El banco de pruebas se articula en torno a tres componentes técnicos principales. Mininet proporciona la emulación de *hosts*, enlaces y *switches* virtuales; Open vSwitch implementa el *datapath* OpenFlow sobre el que se instalan las reglas de *forwarding*; y OpenDaylight actúa como controladora SDN remota, permitiendo descubrir nodos, consultar inventario y programar el plano de datos.

Cada componente cumple una función diferenciada dentro del entorno experimental. Mininet instancia la red emulada, Open vSwitch ejecuta las reglas en el plano de datos y OpenDaylight proporciona el plano de control. La Tabla 3.2 sintetiza esta separación funcional.

Tabla 3.2: Componentes principales del entorno experimental y función dentro del banco de pruebas

Componente	**Función**	**Papel en el banco de pruebas**
Ubuntu / Linux	Plataforma de ejecución del entorno experimental	Sistema base sobre el que se instalan y ejecutan Mininet, Open vSwitch, OpenDaylight y los scripts de automatización.
Mininet	Emulación de topologías de red	Permite desplegar de forma reproducible las topologías utilizadas en el TFG, desde escenarios simples de validación hasta arquitecturas más complejas.
Open vSwitch	Implementación software del *switch* OpenFlow	Actúa como *datapath* de los *switches* virtuales, ejecutando las reglas de *forwarding* instaladas desde la controladora SDN.
OpenDaylight	Controladora SDN remota	Gestiona el descubrimiento de nodos e inventario, y permite programar y supervisar el plano de datos mediante OpenFlow y RESTCONF.
Python	Lenguaje prioritario de automatización	Se utiliza para la generación de topologías, el apoyo a la automatización, el procesado de resultados y la creación de utilidades y figuras.
Bash	Automatización auxiliar del entorno	Se emplea como apoyo operativo en tareas de arranque, limpieza, ejecución de comandos y orquestación sencilla del banco de pruebas.
Overleaf / LaTeX	Redacción y maquetación académica	Se utiliza para integrar en la memoria final del TFG el contenido redactado, así como las tablas, figuras y evidencias generadas.

A partir de estos componentes se define una arquitectura lógica en la que la automatización actúa como capa superior de operación, OpenDaylight constituye el plano de control, y Mininet junto con Open vSwitch forman el plano de datos emulado. Esta arquitectura separa entre automatización, control, datos y trazabilidad de evidencias, como se ilustra en la Figura 3.1.

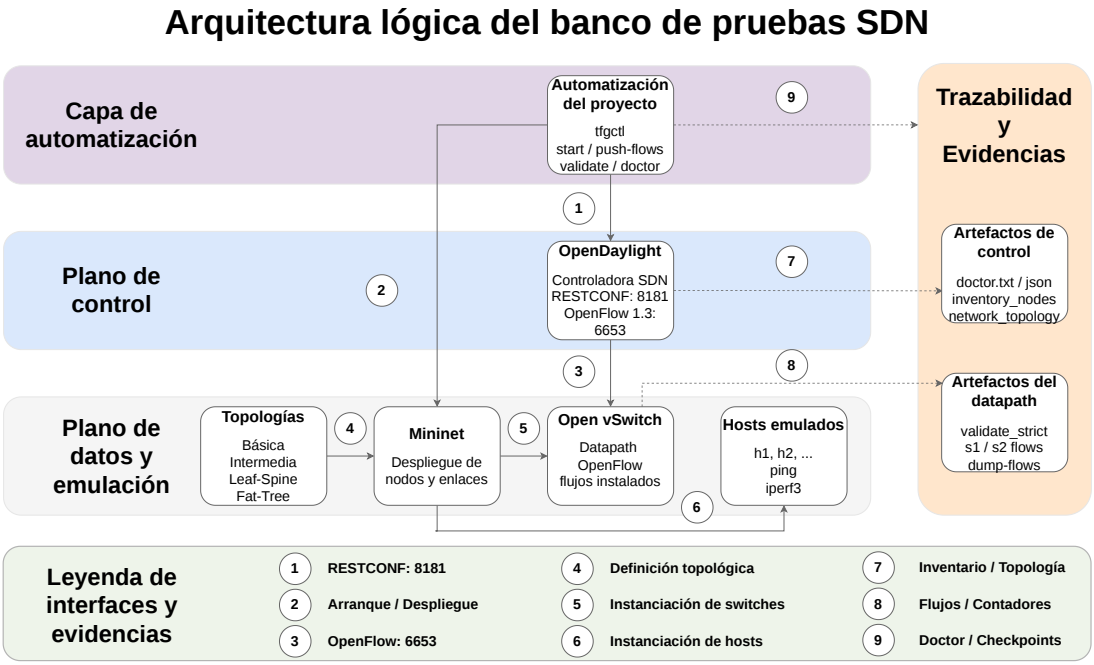

Figura 3.1: Arquitectura lógica del banco de pruebas SDN.

Esta separación permite mantener una distinción clara entre los distintos niveles del sistema. La topología se define y despliega en Mininet, el *datapath* se materializa en Open vSwitch, la controladora OpenDaylight mantiene la visión lógica de nodos y topología, y la capa de automatización coordina el flujo operativo del banco de pruebas. Además, las validaciones generan artefactos persistentes tanto del plano de control como del *datapath*, reforzando la trazabilidad experimental del proceso.

3.1.3. Justificación del entorno finalmente utilizado

La elección de este entorno responde a tres criterios principales. En primer lugar, permite emular topologías de red de forma flexible y controlada, lo que resulta adecuado para avanzar desde escenarios simples hasta arquitecturas redundantes y posteriormente topologías Fat-Tree. En segundo lugar, facilita la separación entre plano de control y plano de datos, característica central en SDN y necesaria para estudiar la instalación de reglas de *forwarding* desde una controladora. En tercer lugar, proporciona mecanismos de inspección suficientemente detallados para validar el estado del banco mediante inventario, topología, *dumps* de *flows*, contadores y *snapshots* del entorno.

Esta aproximación es coherente con el alcance del TFG: construir un banco de pruebas reproducible y defendible sobre el que estudiar topologías SDN de forma incremental. El objetivo no es simular todos los efectos de una infraestructura física de producción, sino disponer de un entorno académico controlado que permita razonar sobre conectividad, *forwarding*, redundancia, validación y generación de evidencias.

3.2. Arquitectura lógica del banco de pruebas

3.2.1. Controladora remota y switches OpenFlow

La arquitectura empleada sigue el modelo habitual de redes definidas por software: la controladora OpenDaylight se ejecuta como entidad remota y los *switches* virtuales Open vSwitch se conectan a ella mediante OpenFlow 1.3. En este esquema, los *switches* no se consideran elementos aislados, sino parte de un banco de pruebas controlado mediante una capa de automatización que permite arrancar topologías, instalar reglas, generar tráfico y validar el estado final del sistema.

Para considerar el banco de pruebas en estado válido no basta con que Mininet arranque ni con que la controladora esté activa. Es necesario verificar una cadena completa de condiciones: OpenDaylight debe estar operativo, los *bridges* de Open vSwitch deben estar conectados a la controladora, el inventario debe mostrar los nodos OpenFlow esperados, la topología debe ser visible, los *flows* deben estar instalados en el *datapath* y los contadores deben reflejar tráfico real.

El estado verde del banco de pruebas se alcanzó en las dos topologías base utilizadas durante la validación inicial. La comprobación incluyó la operatividad de la controladora, el despliegue correcto en Mininet, la conexión de Open vSwitch, la programación del *datapath* y la validación estricta final, tal como se recoge en la Tabla 3.3.

Tabla 3.3: Resumen del estado verde del banco de pruebas en las topologías básica e intermedia

Comprobación	**Básica**	**Intermedia**	**Evidencia resumida**
OpenDaylight operativo	PASS	PASS	Controladora arrancada mediante Karaf, con servicios RESTCONF y OpenFlow activos.
Mininet desplegado correctamente	PASS	PASS	Topologías levantadas mediante `mn -custom ... -topo basic/intermediate` sobre las definiciones Python del proyecto.
Open vSwitch conectado a la controladora	PASS	PASS	*Bridges* OVS conectados a OpenDaylight mediante OpenFlow 1.3 y configurados con `fail-mode=secure`.
push-flows ejecutado correctamente	PASS	PASS	Programación del *forwarding* aplicada desde el *wrapper* del proyecto mediante `tfgctl push-flows`.
Tráfico real generado	PASS	PASS	Ejecución de `pingall` con 0 % de pérdidas en las topologías básica e intermedia.
Inventario y topología visibles en OpenDaylight	PASS	PASS	Validación de inventario OpenFlow y de la topología operacional descubierta por la controladora.
Flows presentes en *datapath*	PASS	PASS	Reglas visibles en el *datapath* de Open vSwitch y conservadas como evidencia en los *checkpoints*.
Validación estricta final	PASS+	PASS+	`tfgctl validate -strict` confirmó ODL operativo, OVS conectado, nodos/topología visibles, *flows* instalados y contadores positivos tras tráfico real.

La validación estricta proporciona una evidencia compacta de que el banco de pruebas no se encontraba solo arrancado, sino operativo desde la perspectiva del controlador y del *datapath*. En las topologías básica e intermedia se confirmó la visibilidad del inventario

y de la topología en OpenDaylight, así como la existencia de *flows* activos con contadores positivos. El Código 3.1 conserva un extracto representativo de esa validación.

```
BASICA:
Nodes: openflow:1
OK: Topology incluye flow:1
OK: s1: counters >0 (trafico confirmado usando flows)
PASS+: ODL listening + OVS connected + ODL sees nodes/topology + flows
in datapath + counters >0.

INTERMEDIA:
Nodes: openflow:1 openflow:2
OK: Topology incluye flow:1
OK: s1: counters >0 (trafico confirmado usando flows)
OK: s2: counters >0 (trafico confirmado usando flows)
PASS+: ODL listening + OVS connected + ODL sees nodes/topology + flows
in datapath + counters >0.
```

Código 3.1: Extracto representativo de la validación estricta del banco de pruebas en las topologías básica e intermedia.

Además del cierre formal con PASS+, se conservó evidencia de bajo nivel del *datapath*. Los *dumps* de Open vSwitch permitieron comprobar que las reglas instaladas recibían tráfico real y que sus contadores evolucionaban de forma positiva durante las pruebas. El Código 3.2 muestra un extracto de esa evidencia.

```
BASICA - s1:
table=0, priority=300, arp actions=NORMAL
    n_packets>0, n_bytes>0
table=0, priority=200, ip,nw_dst=10.0.0.2 actions=output:2
    n_packets>0, n_bytes>0
table=0, priority=0 actions=CONTROLLER:65535
    n_packets>0, n_bytes>0

INTERMEDIA - s1:
table=0, priority=300, arp actions=NORMAL
    n_packets>0, n_bytes>0
table=0, priority=200, ip,nw_dst=10.0.0.3 actions=output:3
    n_packets>0, n_bytes>0

INTERMEDIA - s2:
table=0, priority=300, arp actions=NORMAL
    n_packets>0, n_bytes>0
table=0, priority=200, ip,nw_dst=10.0.0.1 actions=output:2
    n_packets>0, n_bytes>0
```

Código 3.2: Extracto representativo de los *flows* instalados en Open vSwitch y de sus contadores positivos tras la generación de tráfico real.

Esta combinación de evidencias permite justificar que el banco de pruebas no se validó únicamente mediante conectividad superficial, sino mediante una comprobación conjunta de controladora, inventario, topología, *datapath* y tráfico.

3.2.2. Pipeline de validación, captura y procesado

Para evitar que cada ejecución dependiera de pasos manuales difíciles de reproducir, se definió un *pipeline* operativo común. Este *pipeline* organiza la ejecución del banco de pruebas desde el arranque de la controladora hasta la generación del *checkpoint* de evidencias.

El flujo seguido parte del arranque de OpenDaylight y del despliegue de la topología en Mininet. A continuación, se configura Open vSwitch en modo seguro, se instala el *forwarding*, se genera tráfico real y se ejecuta una validación estricta. Finalmente, las salidas relevantes se conservan en un directorio de *checkpoint* asociado a la ejecución. La Figura 3.2 recoge esta secuencia operativa y la relación entre validación, tráfico y conservación de evidencias.

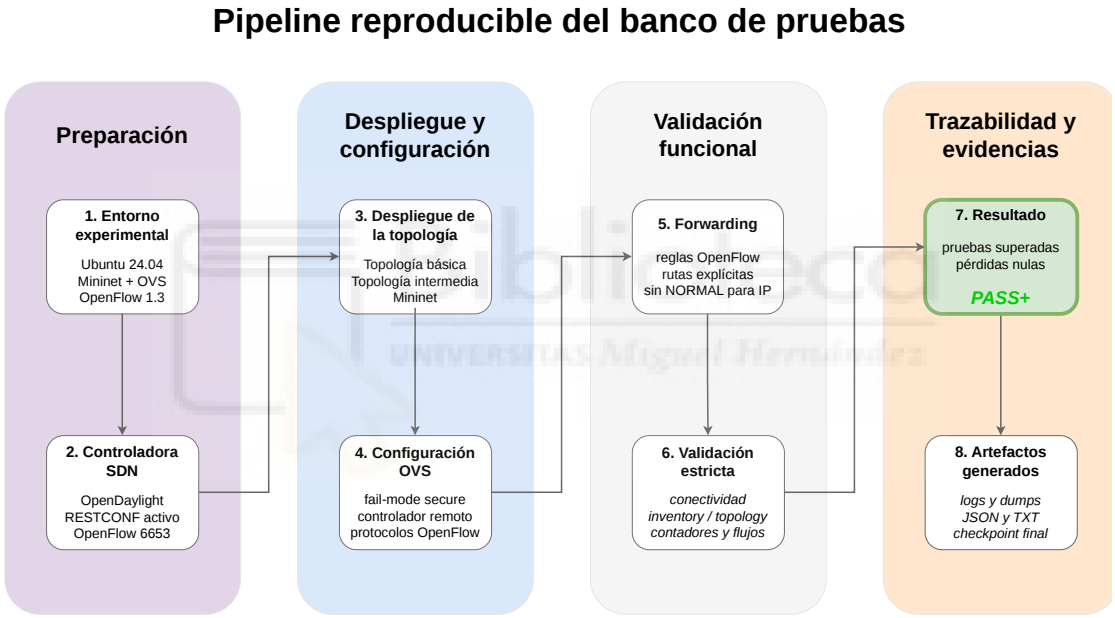

Figura 3.2: Pipeline reproducible del banco de pruebas utilizado en las topologías básica e intermedia.

El enfoque anterior permite separar claramente la ejecución experimental de la interpretación posterior de resultados. Primero se garantiza que el entorno se encuentra en estado válido; después, sobre esa base, pueden ejecutarse campañas o comparaciones más complejas.

3.2.3. Reproducibilidad: versiones, snapshots y estructura del proyecto

La reproducibilidad del banco de pruebas no depende únicamente de los comandos ejecutados, sino también de la organización del proyecto y de la conservación de evidencias. Por este motivo, el repositorio se estructuró separando código, configuración, datos, figuras, documentación y salidas de ejecución.

La estructura del proyecto facilita la trazabilidad entre los *scripts* utilizados, las evidencias generadas y la redacción final de la memoria. La separación entre configuración, código, documentación, figuras, datos y *checkpoints* evita mezclar artefactos de distinta naturaleza y permite reconstruir el origen de cada resultado integrado en el documento. La Tabla 3.4 detalla esta organización.

Tabla 3.4: Estructura principal del proyecto utilizada para organizar el banco de pruebas

Directorio	Contenido principal	Papel en el banco de pruebas
`config/`	Ficheros de configuración del entorno y parámetros auxiliares del proyecto.	Agrupa información reutilizable para mantener separada la configuración de la lógica de ejecución.
`data/`	Datos generados o procesados durante las campañas experimentales.	Permite separar resultados brutos y procesados cuando se ejecutan experimentos o análisis posteriores.
`docs/`	Tablas, fragmentos LaTeX, notas técnicas y material de apoyo para la memoria.	Centraliza evidencias documentales que después se integran en Overleaf o en los capítulos del TFG.
`figures/`	Figuras finales exportadas para la memoria.	Contiene diagramas, gráficas y material visual empleado como evidencia en la redacción académica.
`scripts/`	Scripts auxiliares de operación o apoyo.	Reúne utilidades complementarias que no forman parte directa del paquete Python principal.
`tfg_sdn/`	Paquete Python principal del proyecto. Incluye CLI, topologías, validación, utilidades y módulos de automatización.	Implementa la lógica central del banco de pruebas reproducible y articula la interacción con Mininet, OVS y OpenDaylight.
`runs/`	Directorios de ejecución y *checkpoints*, como `.../checkpoint_basic_ok` o `.../checkpoint_intermediate_ok`.	Conserva evidencias de validación, *dumps* de *datapath*, inventario, topología y *snapshots* asociados a ejecuciones concretas.

Además de registrar las versiones principales del entorno, el banco de pruebas incorpora *snapshots* generados mediante `tfgctl doctor`. Estos *snapshots* conservan información sobre el sistema operativo, *kernel*, versión de Python, versiones de Open vSwitch y Mininet, configuración de Java y puertos asociados a OpenDaylight. La Tabla 3.5 incluye un extracto representativo para las topologías básica e intermedia.

Tabla 3.5: Extracto representativo del *snapshot* del entorno experimental generado mediante `tfgctl doctor`

Elemento verificado	**Topología básica**	**Topología intermedia**
Timestamp del *snapshot*	`2026-04-24T18:41:11+02:00`	`2026-04-24T17:58:25+02:00`
Raíz del repositorio	`/home/narvalaina/Escritorio/TFG/TFG-SDN`	`/home/narvalaina/Escritorio/TFG/TFG-SDN`
Sistema operativo	Ubuntu 24.04.4 LTS	Ubuntu 24.04.4 LTS
Kernel	`6.17.0-22-generic`	`6.17.0-22-generic`
Python	`3.12.3`	`3.12.3`
Open vSwitch	`ovs-vsctl 3.3.4 / ovs-ofctl 3.3.4`	`ovs-vsctl 3.3.4 / ovs-ofctl 3.3.4`
Mininet	`2.3.0`	`2.3.0`
Java	OpenJDK 21.0.10	OpenJDK 21.0.10
OpenDaylight RESTCONF	`127.0.0.1:8181`	`127.0.0.1:8181`
OpenDaylight OpenFlow	`127.0.0.1:6653`	`127.0.0.1:6653`
Memoria disponible	Aproximadamente 10 GiB disponibles	Aproximadamente 10 GiB disponibles
Estado de red asociado	Interfaces de `s1` presentes durante el *snapshot*.	Interfaces de `s1` y `s2` presentes durante el *snapshot*.

De este modo, cada cierre de validación puede asociarse a un estado concreto del entorno experimental. Esta información resulta especialmente útil para diferenciar errores debidos a la lógica del experimento de problemas derivados de versiones, servicios no iniciados o configuración incompleta del sistema.

3.3. Automatización implementada

3.3.1. Arranque, limpieza y validación

La automatización del banco de pruebas se centralizó en torno al *wrapper* `tfgctl` y al paquete Python `tfg_sdn`. Esta capa evita depender exclusivamente de comandos manuales aislados y permite mantener una interfaz común para tareas como despliegue, limpieza, validación, empuje de *flows* y generación de *snapshots*.

El banco de pruebas se articula mediante utilidades diferenciadas: el *wrapper* ejecutable del proyecto, la lógica central del CLI, las definiciones topológicas y las utilidades de *snapshot* y reproducibilidad. La Tabla 3.6 recoge esos elementos operativos.

Tabla 3.6: Scripts y utilidades operativas principales del banco de pruebas

Script / utilidad	Ruta	Función principal	Papel en el banco de pruebas
`tfgctl`	Raíz del proyecto	*Wrapper* ejecutable del proyecto.	Punto de entrada operativo para subcomandos como `push-flows`, `validate` y `doctor`.
`cli.py`	`tfg_sdn/cli.py`	Implementación del CLI principal.	Centraliza la lógica de validación, empuje de *flows*, ejecución de utilidades y gestión operativa del banco de pruebas.
`runner.py`	`tfg_sdn/mininet/runner.py`	*Runner* de Mininet.	Da soporte al despliegue y ejecución de topologías dentro de la infraestructura Python del proyecto.
`basic.py`	`tfg_sdn/mininet/topos/basic.py`	Definición de la topología básica.	Primer escalón reproducible de validación incremental del banco de pruebas.
`intermediate.py`	`tfg_sdn/mininet/topos/intermediate.py`	Definición de la topología intermedia.	Segundo escalón reproducible de validación incremental, ya validado antes del salto a topologías más complejas.
`doctor.py`	`tfg_sdn/doctor.py`	*Snapshot* y verificación del entorno.	Soporte de reproducibilidad y trazabilidad mediante la captura del estado del entorno experimental.

El comando de validación se diseñó para comprobar distintos niveles del sistema: escucha de OpenDaylight en los puertos RESTCONF y OpenFlow, presencia de *bridges* OVS, configuración de `fail-mode`, conexión con la controladora, visibilidad en inventario, visibilidad de la topología, presencia de *flows* y actividad de contadores cuando se ejecuta en modo estricto. Esta validación evita aceptar como correcto un entorno que únicamente haya arrancado, pero que no tenga *datapath* activo o que no haya procesado tráfico real.

3.3.2. Empuje de flows y perfiles de forwarding

Una parte esencial del banco de pruebas es la separación entre la definición de la topología y la política de *forwarding* aplicada sobre ella. Esta separación permite usar una misma topología con distintos perfiles de *forwarding*, o introducir perfiles más complejos cuando se incorporan arquitecturas redundantes.

Los perfiles implementados cubren desde reglas estáticas hasta mecanismos específicos para escenarios con caminos alternativos. Cada perfil se asocia a un mecanismo de instalación concreto, de forma que el capítulo diferencia explícitamente entre reglas instaladas mediante RESTCONF/OpenDaylight y programación directa sobre OVS cuando se requieren grupos OpenFlow `select`. Esta diferenciación se sintetiza en la Tabla 3.7.

Tabla 3.7: Perfiles de *forwarding* implementados en el banco de pruebas

Perfil	**Topología asociada**	**Mecanismo de instalación**	**Función dentro del banco de pruebas**
`basic_l3_static`	Topología básica	RESTCONF mediante `odl.put_flow(...)`	Instala reglas IPv4 por destino en `openflow:1`, junto con ARP y *table-miss*, para validar el funcionamiento elemental del banco de pruebas.
`intermediate_l3_static`	Topología intermedia	RESTCONF mediante `odl.put_flow(...)`	Extiende el *forwarding* estático a dos *switches* OpenFlow, permitiendo validar conectividad, *datapath* y contadores en una estructura más compleja.
`leafspine_l3_static`	Leaf-Spine	RESTCONF mediante `odl.put_flow(...)`	Implementa un reparto estático por destino, usado como *baseline* metodológico frente a mecanismos *multipath* más generales.
`leafspine_single_path`	Leaf-Spine	RESTCONF mediante `odl.put_flow(...)`	Fuerza el tráfico remoto a utilizar un único *uplink*, permitiendo observar el efecto de desaprovechar la redundancia disponible.
`leafspine_ecmp`	Leaf-Spine	Programación directa en OVS mediante `ovs-ofctl`.	Instala reglas y grupos OpenFlow de tipo `select` para repartir tráfico entre caminos equivalentes en el *fabric* Leaf-Spine.

En las topologías básica e intermedia, así como en los perfiles estáticos de Leaf-Spine, la instalación de reglas se realiza mediante OpenDaylight y RESTCONF. El proyecto construye reglas de *forwarding* como estructuras Python/JSON y las envía a la controladora para su instalación en la tabla correspondiente del nodo OpenFlow.

El mecanismo de instalación RESTCONF se ilustra mediante una regla IPv4 por destino, generada desde Python y enviada a OpenDaylight para programar el nodo OpenFlow correspondiente. El Código 3.3 permite identificar la estructura de la regla, los campos de coincidencia y la acción de salida asociada.

```
# Ejemplo representativo: regla IPv4 por destino hacia un puerto de salida

# 1) Construccion de la regla en flows.py
flow = flow_ipv4_dst_to_port(
    flow_id="to_h1_ip",
    ipv4_dst="10.0.0.1/32",
    out_port="1"
)

# 2) Estructura logica del flow generado
flow = {
    "id": "to_h1_ip",
    "table_id": 0,
    "priority": 200,
    "match": {
        "ethernet-type": 2048,
        "ipv4-destination": "10.0.0.1/32"
    },
    "instructions": {
        "output-node-connector": "1"
    }
}

# 3) Ruta RESTCONF generada por el cliente ODL
/rests/data/opendaylight-inventory:nodes/node=openflow:1
/flow-node-inventory:table=0/flow=to_h1_ip

# 4) Payload enviado a OpenDaylight
{
    "flow-node-inventory:flow": [flow]
}

# 5) Instalacion desde el CLI del proyecto
code, body = odl.put_flow(
    node="openflow:1",
    table_id=0,
    flow=flow
)
```

Código 3.3: Extracto representativo de la instalación de una regla de *forwarding* mediante RESTCONF. La regla corresponde a un *flow* IPv4 por destino instalado en `openflow:1` con prioridad 200 y acción de salida hacia un puerto concreto.

El mecanismo anterior se emplea para instalar *flows* estáticos, por ejemplo reglas IPv4 por destino hacia un puerto de salida determinado, junto con reglas auxiliares como ARP y *table-miss*. La evidencia de *datapath* mostrada anteriormente confirma que estas reglas no solo fueron enviadas a la controladora, sino que terminaron materializándose en Open vSwitch.

En el perfil ECMP de Leaf-Spine se utilizó una aproximación específica basada en grupos OpenFlow de tipo `select`. En este caso, la programación se realiza directamente sobre OVS mediante `ovs-ofctl`, ya que esta vía permite manipular explícitamente *flows* y grupos OpenFlow en el *datapath*. Las rutas remotas se asocian a una acción `group:100`, como se observa en el Código 3.4.

```
# Ejemplo representativo: grupo OpenFlow select para ECMP

LEAFSPINE_REMOTE_GID = 100

# Construccion del grupo multipath
group_spec = (
    "group_id=100,type=select,"
    "bucket=output:1,bucket=output:2"
)

# Instalacion del grupo en cada leaf
_ofctl_add_group(br, group_spec)

# Uso del grupo por las rutas remotas
table=0,priority=200,ip,nw_dst=10.0.0.X actions=group:100

# Validacion posterior
dump-groups
dump-group-stats
```

Código 3.4: Extracto representativo del uso de grupos OpenFlow de tipo `select` en el perfil Leaf-Spine ECMP. El grupo agrupa varios *buckets* de salida hacia los *spines* y permite que las rutas remotas utilicen una acción `group:100` en lugar de fijar un único puerto de salida.

Esta distinción es importante desde el punto de vista metodológico. Los perfiles estáticos se instalan mediante RESTCONF y OpenDaylight, mientras que el perfil ECMP utiliza programación directa en OVS para controlar explícitamente los grupos `select`. No se trata de una contradicción, sino de una decisión técnica documentada: el banco de pruebas combina ambos mecanismos cuando cada uno resulta más adecuado para el tipo de *forwarding* que se desea instalar.

3.3.3. Ejecución de experimentos y generación de evidencias

El *pipeline* del banco de pruebas genera un conjunto de artefactos que permiten reconstruir el estado de cada validación. Estos artefactos evitan depender únicamente de la observación en consola y permiten conservar trazas de inventario, topología, *datapath* y entorno.

Los artefactos generados relacionan cada ejecución con su validación estricta, el estado del *datapath*, la vista operacional de OpenDaylight y el *snapshot* del entorno experimental. La Tabla 3.8 recoge la función de cada tipo de archivo dentro del *pipeline*.

Tabla 3.8: Artefactos generados por el *pipeline* de validación del banco de pruebas

Artefacto	Contenido	Topología	Utilidad en el banco de pruebas
`validate_strict.txt`	Salida completa de `./tfgctl validate -topo ... -strict`.	Básica e intermedia	Documenta el cierre formal de validación fuerte, incluyendo ODL operativo, OVS conectado, inventario/topología visibles, *flows* en *datapath* y contadores positivos.
`s1_flows.txt`	Volcado de *flows* de Open vSwitch en `s1` mediante `ovs-ofctl`.	Básica e intermedia	Permite comprobar que las reglas OpenFlow fueron instaladas en *datapath* y que sus contadores registraron tráfico real.
`s2_flows.txt`	Volcado de *flows* de Open vSwitch en `s2`.	Intermedia	Extiende la comprobación del *datapath* al segundo *switch* de la topología intermedia.
`inventory_nodes.json`	Respuesta RESTCONF del inventario operacional de OpenDaylight.	Básica e intermedia	Verifica que la controladora descubre los nodos OpenFlow esperados.
`topology_flow1.json`	Respuesta RESTCONF de la topología operacional `flow:1`.	Básica e intermedia	Confirma que OpenDaylight mantiene una vista operacional de la topología descubierta.
`doctor.txt` / `doctor.json`	*Snapshot* del estado del entorno experimental generado mediante `./tfgctl doctor`.	Básica e intermedia	Refuerza la trazabilidad del entorno y permite conservar una referencia del estado software y operativo asociado a la ejecución.
`runs/.../checkpoint_...`	Directorio de *checkpoint* con los artefactos de una validación concreta.	Básica e intermedia	Agrupa las evidencias de cada cierre experimental y evita mezclar resultados de ejecuciones distintas.

En conjunto, estos ficheros forman el cierre documental de cada validación. El archivo `validate_strict.txt` recoge el resultado formal de la validación; los ficheros `s1_flows.txt` y `s2_flows.txt` permiten inspeccionar el *datapath*; los JSON de inventario y topología conservan la visión operacional de OpenDaylight; y los ficheros `doctor` registran el estado del entorno.

3.4. Lecciones de implementación

3.4.1. Problemas iniciales con flooding ARP y conectividad

Durante la construcción del banco de pruebas aparecieron problemas propios de un entorno SDN emulado: validaciones ejecutadas demasiado pronto, ausencia de tráfico real antes de validar, errores derivados de privilegios insuficientes, comandos ejecutados desde contextos no adecuados y problemas de registro de topologías en Mininet. Estos proble-

mas no deben interpretarse como incidencias aisladas, sino como parte del proceso de estabilización del banco de pruebas.

La consolidación del entorno exigió identificar y resolver problemas que podían generar validaciones prematuras, errores de interpretación del estado de Open vSwitch o ejecuciones no reproducibles. La Tabla 3.9 recoge esas incidencias junto con su efecto metodológico dentro del proceso de estabilización.

Tabla 3.9: Problemas iniciales detectados durante la consolidación del banco de pruebas

Problema detectado	**Síntoma observado**	**Impacto experimental**	**Corrección adoptada**
Validación estricta prematura	`validate -strict` podía fallar si aún no había *flows* instalados o tráfico real.	No se podían confirmar *datapath* activo ni contadores positivos.	Fijar el orden: instalación de *flows*, generación de tráfico y validación estricta.
Ausencia de tráfico previo	Los *flows* existían, pero sin contadores positivos en el *datapath*.	No se podía demostrar que las reglas hubieran sido usadas por tráfico real.	Ejecutar `pingall` o tráfico equivalente antes de validar.
Privilegios insuficientes en OVS	El CLI podía devolver mensajes confusos sin un *ticket* `sudo` activo.	La inspección de *bridges* OVS podía interpretarse de forma errónea.	Ejecutar `sudo -v` y reforzar el *pre-flight* del CLI.
Comandos OVS en Mininet	Comandos como `ovs-vsctl` fallaban al ejecutarse desde `mininet>`.	La configuración de OVS no se aplicaba desde el contexto correcto.	Ejecutar comandos OVS con `sudo` desde una terminal *bash*.
Topología no registrada	`mn -custom ... -topo basic` devolvía un error de topología inválida.	La topología básica no podía levantarse de forma reproducible.	Añadir el diccionario `topos` al archivo `basic.py`.
Variables ambiguas	Había nombres ambiguos para puertos OpenFlow y RESTCONF.	Podían aparecer configuraciones inconsistentes entre comandos y scripts.	Normalizar `ODL_OF_PORT` y `ODL_REST_PORT`.

Uno de los aprendizajes más relevantes fue que la conectividad aparente no es suficiente para validar un banco de pruebas SDN. En este trabajo se exigió que los *flows* estuvieran instalados, que el *datapath* pudiera inspeccionarse y que los contadores reflejaran tráfico real. Esta decisión reduce la posibilidad de aceptar resultados apoyados en comportamientos implícitos o en estados incompletos del sistema.

3.4.2. Decisiones de diseño adoptadas

A partir de los problemas anteriores se adoptaron varias decisiones de diseño orientadas a reducir ambigüedad, aumentar el control del *datapath* y mejorar la trazabilidad de cada ejecución. Entre las decisiones más importantes se encuentra definir un orden reproducible de ejecución, instalar el *forwarding* antes de validar, generar tráfico real antes de exigir contadores positivos, fijar `fail-mode=secure` en los *bridges* OVS, normalizar variables de entorno y conservar *checkpoints* de evidencia.

La Tabla 3.10 relaciona estas decisiones con su beneficio metodológico dentro del banco de pruebas.

Tabla 3.10: Decisiones de diseño adoptadas para estabilizar el banco de pruebas

Decisión adoptada	**Problema que resuelve**	**Beneficio metodológico**
Definir un orden reproducible de ejecución	Evita validar el entorno antes de instalar *forwarding* o generar tráfico real.	Establece una secuencia verificable: arranque de ODL, despliegue Mininet, configuración OVS, `push-flows`, tráfico real y `validate -strict`.
Ejecutar `push-flows` antes de la validación estricta	Evita que el *datapath* se valide sin reglas esperadas instaladas.	Permite comprobar que el *forwarding* programado está realmente presente en Open vSwitch.
Generar tráfico real antes de `validate -strict`	Evita que los *flows* aparezcan instalados pero sin actividad.	Permite exigir contadores positivos y demostrar que el tráfico ha utilizado las reglas instaladas.
Fijar `fail-mode=secure` en los *bridges* OVS	Reduce comportamientos no controlados del *switch* ante ausencia o fallo de controlador.	Refuerza el control explícito del *datapath* y evita depender de mecanismos implícitos de *forwarding*.
Ejecutar `sudo -v` antes de validar	Evita fallos confusos en consultas OVS que requieren privilegios.	Hace que la validación sea más robusta y que los errores sean más interpretables.
Normalizar variables de entorno ODL	Evita ambigüedad entre puertos OpenFlow y RESTCONF.	Mejora la reproducibilidad de los comandos y la coherencia entre scripts.
Corregir `basic.py` exportando `topos`	Permite que Mininet reconozca la topología básica mediante `mn -custom ... -topo basic`.	Hace que la topología básica sea desplegable de forma reproducible desde el flujo operativo oficial.
Cerrar cada validación con *checkpoint* de evidencias	Evita mezclar salidas de ejecuciones distintas o depender solo de observaciones en consola.	Conserva trazabilidad mediante ficheros como `validate_strict.txt`, `dump-flows`, inventario, topología y *snapshots* del entorno.

Estas decisiones permitieron transformar un conjunto de herramientas en un banco de pruebas reproducible y defendible. La principal conclusión del capítulo es que el banco de pruebas no se limita a desplegar topologías en Mininet, sino que incorpora una metodología de validación e inspección del estado de red. Esta base resulta necesaria antes de avanzar hacia topologías redundantes y hacia el estudio posterior de arquitecturas Fat-Tree multidimensionales.

4. VALIDACIÓN DEL BANCO DE PRUEBAS

4.1. Objetivo y alcance de la validación

Antes de abordar topologías de mayor complejidad, resulta necesario validar que el banco de pruebas implementado funciona de forma correcta, reproducible y trazable. El objetivo de este capítulo no es todavía comparar arquitecturas de red desde el punto de vista experimental, sino comprobar que el entorno formado por Mininet, Open vSwitch, OpenDaylight y las herramientas de automatización desarrolladas permite desplegar topologías, instalar reglas de *forwarding*, generar tráfico, observar el *datapath* y obtener métricas básicas de funcionamiento.

La validación se plantea de forma incremental. En primer lugar, se utiliza una topología básica con un único *switch* OpenFlow y dos *hosts*, con el fin de comprobar la conectividad elemental del entorno. Posteriormente, se emplea una topología intermedia con dos *switches* OpenFlow, cuatro *hosts* y un enlace inter-*switch*, lo que permite verificar el comportamiento del banco de pruebas en un escenario multi-*switch*. Esta estrategia permite detectar problemas de forma progresiva, evitando introducir directamente una topología compleja sin haber validado previamente los componentes fundamentales del entorno.

Por tanto, las evidencias presentadas en este capítulo deben interpretarse como una validación previa del banco de pruebas. Su finalidad es demostrar que el entorno está preparado para las campañas posteriores sobre Leaf-Spine y Fat-Tree, no extraer todavía conclusiones definitivas sobre el rendimiento relativo de distintas arquitecturas.

4.2. Topologías empleadas en la validación incremental

La validación se realizó sobre dos topologías de complejidad creciente. La primera corresponde a una topología básica formada por dos *hosts*, `h1` y `h2`, conectados a un único *switch* OpenFlow, `s1`. Esta topología permite validar el caso mínimo de conectividad extremo a extremo bajo control SDN.

La segunda corresponde a una topología intermedia formada por cuatro *hosts* y dos *switches* OpenFlow. En esta topología, `h1` y `h3` se conectan a `s1`, mientras que `h2` y `h4` se conectan a `s2`. Ambos *switches* están unidos mediante un enlace inter-*switch*. Esta configuración permite comprobar no solo la conectividad entre *hosts* conectados a un mismo entorno de conmutación, sino también el tráfico que debe atravesar un enlace entre *switches*.

La comparación visual de ambas topologías refuerza la lógica incremental de la validación. El escenario básico sirve como punto de partida para comprobar la comunicación mínima

entre dos *hosts*, mientras que el escenario intermedio introduce un segundo *switch* y un enlace entre conmutadores para validar reglas, conectividad, contadores y observabilidad del tráfico en un entorno ligeramente más complejo. Esta progresión se representa en la Figura 4.1.

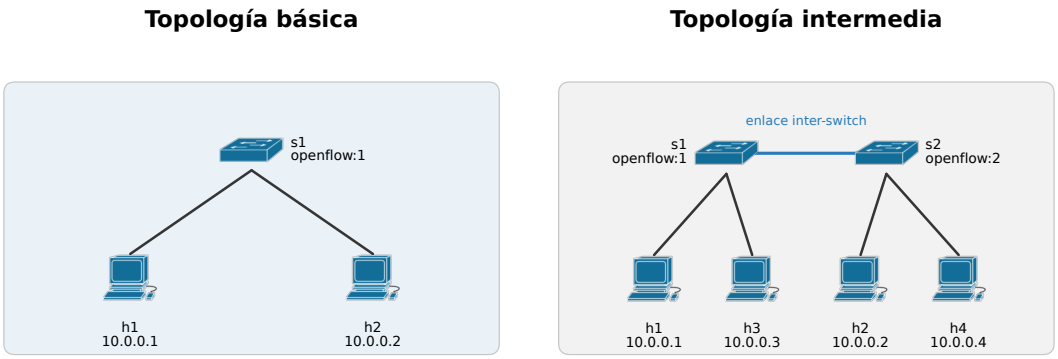

Figura 4.1: Topologías empleadas para la validación incremental del banco de pruebas.

4.3. Criterios de validación

Para evitar que la validación se limite a una comprobación puntual de conectividad, se definieron varios criterios de aceptación. Estos criterios cubren tanto el plano de control como el plano de datos, así como la generación de evidencias reproducibles. En concreto, se comprobó el arranque de la controladora, la conexión de los *switches* OpenFlow, la presencia de nodos en el inventario de OpenDaylight, la instalación explícita de *flows*, la conectividad entre *hosts*, la presencia de contadores positivos en Open vSwitch, la observación directa de tráfico ARP/ICMP y la generación de métricas cuantitativas mínimas.

La definición previa de estos criterios establece qué significa considerar validado el banco de pruebas. La conectividad ICMP es necesaria, pero no suficiente por sí sola: también es necesario comprobar que la controladora está disponible, que los *switches* están conectados, que las reglas aparecen en el *datapath* y que el tráfico puede observarse y medirse.

Los criterios utilizados durante esta fase se recogen en la Tabla 4.1.

Tabla 4.1: Criterios utilizados para la validación incremental del banco de pruebas

Criterio	**Qué comprueba**	**Evidencia asociada**
Despliegue de la topología	La topología se inicia correctamente en Mininet y aparecen los *hosts*, *switches* y enlaces esperados.	Figura de topología, salida de arranque y comprobación de nodos.
Conectividad extremo a extremo	Los *hosts* pueden comunicarse entre sí mediante tráfico ICMP en las pruebas iniciales de conectividad.	Resultado de `pingall` y pruebas entre *hosts*.
Visibilidad desde OpenDaylight	La controladora detecta los nodos OpenFlow esperados y mantiene una vista operacional de la topología.	Inventario operacional y topología descubierta por OpenDaylight.
Instalación de *forwarding*	Las reglas necesarias para la comunicación se instalan sobre los *switches* OpenFlow correspondientes.	Ejecución de `tfgctl push-flows` y confirmación de reglas instaladas.
Validación del *datapath*	Los *flows* aparecen en Open vSwitch y sus contadores reflejan actividad tras generar tráfico real.	`dump-flows`, contadores positivos y validación estricta.
Observabilidad del tráfico	El entorno permite capturar o resumir tráfico real asociado a las pruebas de conectividad.	Tráfico ARP/ICMP observado y registros de validación.
Trazabilidad de la ejecución	Las salidas relevantes quedan asociadas a una ejecución concreta y pueden conservarse para revisión posterior.	Directorios `runs/`, *checkpoints*, *snapshots* y ficheros de evidencia.

4.4. Validación de conectividad inicial

Una vez desplegadas las topologías y aplicados los perfiles de *forwarding* correspondientes, se ejecutaron pruebas iniciales de conectividad mediante `pingall`. Esta prueba permite verificar que todos los *hosts* pueden comunicarse entre sí según la topología desplegada. En la topología básica, la prueba se reduce a la comunicación entre `h1` y `h2`. En la topología intermedia, la prueba implica doce comunicaciones ICMP entre los cuatro *hosts*.

Los resultados muestran que ambas topologías alcanzaron conectividad completa. La topología básica obtuvo `0% dropped` con `2/2 received`, mientras que la topología intermedia obtuvo `0% dropped` con `12/12 received`. Estos valores, recogidos en la Tabla 4.2, confirman que tras el arranque de OpenDaylight, el despliegue de Mininet y la instalación de reglas, el banco de pruebas permite comunicación extremo a extremo en escenarios de distinta complejidad.

Tabla 4.2: Resumen de conectividad inicial mediante `pingall` en las topologías básica e intermedia

Topología	Escenario validado	Resultado	Interpretación
Básica	Comunicación entre `h1` y `h2` a través de un único *switch* OpenFlow, `s1`.	`0% dropped 2/2 received`	La conectividad elemental queda validada tras el arranque de OpenDaylight, el despliegue de Mininet y la instalación de los *flows* básicos.
Intermedia	Comunicación completa entre `h1`, `h2`, `h3` y `h4` sobre dos *switches* OpenFlow conectados entre sí.	`0% dropped 12/12 received`	La conectividad multi-*switch* queda validada, incluyendo tráfico local y tráfico que atraviesa el enlace inter-*switch*.

No obstante, esta comprobación inicial no se interpreta como una validación completa del entorno. El resultado de `pingall` demuestra conectividad funcional, pero no verifica por sí solo que los *flows* hayan sido instalados de forma explícita, que el *datapath* refleje contadores positivos o que el tráfico pueda observarse directamente. Por ello, las siguientes secciones profundizan en la instalación de reglas, el estado de Open vSwitch y la observabilidad del tráfico.

4.5. Inserción de reglas de *forwarding*

La conectividad validada en la sección anterior no se dejó al comportamiento automático de Mininet, sino que se obtuvo mediante la instalación explícita de reglas de *forwarding*. Para ello se utilizó la herramienta `tfgctl push-flows`, integrada dentro del flujo operativo del proyecto. Esta herramienta permite instalar reglas en los *switches* OpenFlow a través de OpenDaylight y RESTCONF, manteniendo una separación clara entre el despliegue de la topología y la programación del plano de datos.

La instalación de reglas se adaptó a la complejidad de cada topología. En la topología básica, las reglas se instalan sobre `openflow:1` e incluyen tratamiento ARP, regla *table-miss* y rutas IPv4 hacia `h1` y `h2`. En la topología intermedia, las reglas se instalan tanto en `openflow:1` como en `openflow:2`, diferenciando rutas locales hacia los *hosts* conectados directamente a cada *switch* y rutas remotas hacia los *hosts* alcanzables a través del enlace inter-*switch*. Esta síntesis aparece en la Tabla 4.3.

Tabla 4.3: Resumen de la inserción de flujos en las topologías básica e intermedia

Topología	**Nodo OpenFlow**	***Flows* instalados**	**Función dentro de la validación**

Básica	`openflow:1`	`arp_normal`
		`table_miss`
		`to_h1_ip`
		`to_h2_ip`
			Permiten la comunicación entre `h1` y `h2` a través de un único *switch*. La validación comprueba que el banco de pruebas puede instalar reglas mínimas de conectividad y transportar tráfico ICMP extremo a extremo.

Intermedia	`openflow:1`	`arp_normal`
		`table_miss`
		`to_h1_ip`
		`to_h3_ip`
		`to_h2_ip`
		`to_h4_ip`
			Instalan reglas locales hacia los *hosts* conectados a `s1` y reglas remotas hacia los *hosts* alcanzables a través del enlace inter-*switch*. En la topología real, `h1` y `h3` cuelgan de `s1`.

Intermedia	`openflow:2`	`arp_normal`
		`table_miss`
		`to_h2_ip`
		`to_h4_ip`
		`to_h1_ip`
		`to_h3_ip`
			Instalan reglas locales hacia los *hosts* conectados a `s2` y reglas remotas hacia los *hosts* alcanzables a través del enlace inter-*switch*. En la topología real, `h2` y `h4` cuelgan de `s2`.

El procedimiento de instalación se conserva también mediante salidas representativas de consola. El extracto del Código 4.1 confirma que el inventario de OpenDaylight contiene los nodos esperados antes de iniciar la instalación y que las reglas principales se programan sobre los *switches* correspondientes. Este fragmento no pretende sustituir a los registros completos conservados como evidencia bruta, sino mostrar de forma compacta el procedimiento seguido.

```
BASICA

$ ./tfgctl push-flows --topo basic

==[push-flows] Esperando inventory openflow:1]==
OK: Inventory contiene openflow:1

==[push-flows] Instalando flows BASIC (PUT RESTCONF config datastore)]==
OK: openflow:1 <- arp_normal
OK: openflow:1 <- table_miss
OK: openflow:1 <- to_h1_ip
OK: openflow:1 <- to_h2_ip

DONE: flows BASIC instalados en config (ODL).


INTERMEDIA

$ ./tfgctl push-flows --topo intermediate

==[push-flows] Esperando inventory openflow:1/openflow:2]==
OK: Inventory contiene openflow:1 y openflow:2

==[push-flows] Instalando flows INTERMEDIATE (PUT RESTCONF config datastore)]==
OK: openflow:1 <- arp_normal
OK: openflow:1 <- table_miss
OK: openflow:1 <- to_h1_ip
OK: openflow:1 <- to_h3_ip
OK: openflow:1 <- to_h2_ip
OK: openflow:1 <- to_h4_ip
OK: openflow:2 <- arp_normal
OK: openflow:2 <- table_miss
OK: openflow:2 <- to_h2_ip
OK: openflow:2 <- to_h4_ip
OK: openflow:2 <- to_h1_ip
OK: openflow:2 <- to_h3_ip

DONE: flows INTERMEDIATE instalados en config (ODL).
```

Código 4.1: Extracto representativo de la instalación de *flows* mediante `tfgctl push-flows` en las topologías básica e intermedia.

4.6. Validación del *datapath* y contadores

Una vez instaladas las reglas y generada conectividad mediante `pingall`, se verificó el estado real del *datapath* mediante `ovs-ofctl`. Esta comprobación permite confirmar que los *flows* no solo fueron enviados al entorno de control, sino que aparecen en Open vSwitch y reflejan actividad mediante contadores positivos.

La validación del *datapath* permitió observar reglas ARP, rutas IPv4 y reglas auxiliares coherentes con cada escenario. En la topología básica se comprobaron reglas hacia `10.0.0.1` y `10.0.0.2`, junto con una regla *table-miss*. En la topología intermedia se

observaron reglas locales y remotas en `s1` y `s2`, con contadores positivos tras generar tráfico real. La Tabla 4.4 resume esta comprobación.

Tabla 4.4: Validación de *flows* instalados y contadores positivos en el *datapath*

Topología y *switch*	**Evidencia observada en el *datapath***	**Resultado de validación**
Básica, `s1`	Reglas ARP, rutas IPv4 hacia `10.0.0.1`/`10.0.0.2` y regla *table-miss*, todas ellas con contadores positivos tras la generación de tráfico real.	Validación estricta completada con `PASS+`; el *datapath* procesó tráfico real tras `pingall`.
Intermedia, `s1`	Reglas ARP, rutas locales hacia `10.0.0.1`/`10.0.0.3`, rutas remotas hacia `10.0.0.2`/`10.0.0.4` y regla *table-miss*, con contadores positivos.	`s1` encaminó tráfico local y tráfico dirigido hacia el enlace inter-*switch*.
Intermedia, `s2`	Reglas ARP, rutas locales hacia `10.0.0.2`/`10.0.0.4`, rutas remotas hacia `10.0.0.1`/`10.0.0.3` y regla *table-miss*, con contadores positivos.	`s2` encaminó tráfico local y tráfico dirigido hacia el enlace inter-*switch*.

Esta comprobación resulta especialmente relevante porque permite diferenciar entre una regla configurada y una regla efectivamente activa en el *datapath*. Además, durante esta fase se comprobó la importancia de limpiar el entorno antes de repetir pruebas, evitando que reglas residuales de ejecuciones anteriores pudieran contaminar las evidencias. Por ello, las validaciones finales se realizaron desde un estado controlado y con *dump-flows* coherentes con cada topología.

4.7. Observabilidad del tráfico en el plano de datos

Además de comprobar conectividad y contadores, se capturó tráfico ARP/ICMP mediante `tcpdump` en interfaces del plano de datos. Esta comprobación permite verificar que el tráfico generado durante la validación puede observarse directamente sobre los enlaces emulados, lo que refuerza la capacidad del banco de pruebas para analizar el comportamiento de las topologías.

Las capturas muestran tráfico coherente con cada escenario. En la topología básica, la captura se realizó sobre `s1-eth1`, observando tramas ARP para resolver la dirección de `h2` y paquetes `ICMP echo request`/`ICMP echo reply` entre `h1` y `h2`. En la topología intermedia, la captura se realizó sobre `s1-eth3`, correspondiente al enlace inter-*switch*, observando tráfico ARP e ICMP entre `10.0.0.1` y `10.0.0.2`. La Tabla 4.5 resume estas evidencias.

Tabla 4.5: Resumen de tráfico ARP/ICMP observado durante la validación

Topología	**Punto de captura**	**Tráfico observado**	**Interpretación**
Básica	`s1-eth1`	Tramas ARP para resolver `10.0.0.2` desde `10.0.0.1`, seguidas de paquetes `ICMP echo request` y `ICMP echo reply` entre ambos *hosts*.	La captura confirma que el banco de pruebas permite observar tráfico real asociado a la conectividad básica entre `h1` y `h2`.
Intermedia	`s1-eth3`	Tramas ARP y tráfico ICMP entre `10.0.0.1` y `10.0.0.2`, incluyendo solicitudes y respuestas ICMP capturadas sobre el enlace entre *switches*.	La captura confirma que el tráfico entre *hosts* conectados a *switches* distintos atraviesa el enlace inter-*switch* y puede observarse desde el *datapath*.

Dado que `h1` y `h2` se encuentran conectados a *switches* distintos en la topología intermedia, la captura sobre el enlace inter-*switch* confirma que el tráfico atraviesa efectivamente el enlace entre `s1` y `s2`. Esta evidencia complementa las comprobaciones anteriores: `pingall` demuestra conectividad, los contadores de Open vSwitch demuestran actividad en el *datapath*, y las capturas con `tcpdump` muestran directamente el tráfico que circula por las interfaces emuladas.

4.8. Microvalidación cuantitativa mínima

Como complemento a la validación funcional del banco de pruebas, se realizó una microvalidación cuantitativa mínima sobre las topologías básica e intermedia. El objetivo de esta prueba no fue establecer una comparación experimental exhaustiva entre ambas topologías, sino comprobar que el entorno era capaz de generar métricas de rendimiento coherentes tras el despliegue de la topología, la instalación de *flows* y la verificación de conectividad. Para ello, se midió el *throughput* TCP entre `h1` y `h2` mediante `iperf3`, con tres repeticiones por topología, y se complementó la prueba con una medida de RTT mediante `ping`.

Los resultados de *throughput* muestran que ambas topologías alcanzan valores del mismo orden de magnitud. La topología básica obtuvo un valor medio de 21,73 Gbps, mientras que la topología intermedia alcanzó 20,59 Gbps. Esta diferencia debe interpretarse con prudencia, ya que la prueba se realizó en un entorno emulado local y con un número reducido de repeticiones. La lectura principal no es que una topología sea superior a la otra, sino que el banco de pruebas mantiene capacidad de transporte TCP tanto en el escenario elemental como en el escenario multi-*switch*. Esta comparación se representa en la Figura 4.2.

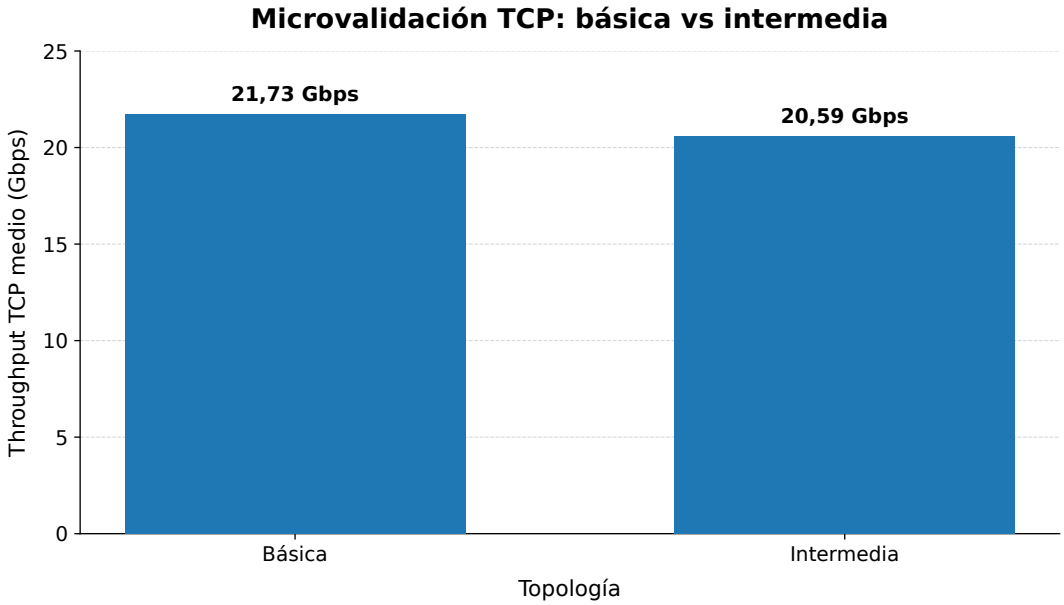

Figura 4.2: Throughput TCP medio en las topologías básica e intermedia.

La latencia RTT observada durante la misma microvalidación también fue coherente con el objetivo de la prueba. En ambos casos se obtuvo 0 % de pérdida ICMP. La topología básica presentó un RTT medio de 0,071 ms, mientras que la topología intermedia presentó un RTT medio de 0,087 ms. El incremento observado es reducido, pero compatible con la presencia de un enlace inter-*switch* adicional en la topología intermedia. Estos valores se recogen en la Tabla 4.6.

Tabla 4.6: Resumen de latencia RTT observada durante la microvalidación cuantitativa

Topología	**Camino validado**	**RTT mín. (ms)**	**RTT med. (ms)**	**RTT máx. (ms)**	**Mdev (ms)**
Básica	`h1` → `h2`, un *switch* OpenFlow	0,035	0,071	0,111	0,022
Intermedia	`h1` → `h2`, enlace inter-*switch*	0,042	0,087	0,117	0,021

En conjunto, esta microvalidación confirma que el banco de pruebas no solo permite desplegar topologías, instalar reglas y validar conectividad, sino que también genera métricas cuantitativas reproducibles de *throughput* y latencia. Estos resultados no se utilizan como comparación experimental definitiva entre topologías, sino como evidencia de que el entorno está preparado para campañas de medida más completas en las fases posteriores del trabajo.

4.9. Lecciones metodológicas de la validación

La validación incremental no solo permitió confirmar el funcionamiento de las topologías básica e intermedia, sino que también permitió identificar condiciones metodológicas necesarias para las siguientes fases del trabajo. Las lecciones extraídas refuerzan la necesidad de trabajar con un flujo experimental controlado: arrancar OpenDaylight antes de desplegar la topología, comprobar la conexión de los *switches* OVS, instalar explícitamente los *flows*, limpiar el entorno antes de repetir pruebas y conservar evidencias trazables. La Tabla 4.7 sintetiza estas conclusiones metodológicas.

Tabla 4.7: Lecciones metodológicas extraídas de la validación incremental del banco de pruebas

Lección metodológica	**Implicación durante la validación**	**Aplicación en fases posteriores**
Validar de forma incremental	Permite detectar problemas primero en una topología mínima y después en un escenario multi-*switch* controlado.	Reduce el riesgo de trasladar errores básicos a Leaf-Spine y Fat-Tree, donde la interpretación sería más compleja.
Arrancar OpenDaylight antes de desplegar Mininet	Garantiza que los *switches* OVS puedan conectarse a la controladora desde el inicio de la ejecución.	Establece una secuencia reproducible de arranque para campañas experimentales posteriores.
Instalar explícitamente los *flows*	Evita depender de comportamientos implícitos o automáticos del entorno de emulación.	Permite comparar perfiles de *forwarding* de forma controlada en arquitecturas redundantes.
Comprobar el *datapath* con contadores	Distingue entre reglas configuradas y reglas realmente activas tras generar tráfico.	Refuerza la validez de las evidencias en experimentos con múltiples *switches* y enlaces.
Conservar evidencias trazables	Relaciona cada validación con tablas, capturas, listados y métricas cuantitativas.	Facilita la reproducibilidad de resultados y la defensa metodológica del TFG.
Limpiar el entorno antes de repetir pruebas	Evita que reglas o estados residuales contaminen ejecuciones posteriores.	Permite ejecutar campañas repetibles con menor ambigüedad experimental.

Estas condiciones serán especialmente importantes en las fases posteriores, donde las topologías Leaf-Spine y Fat-Tree introducen mayor número de *switches*, enlaces y perfiles de *forwarding*. La validación incremental permite, por tanto, avanzar hacia esas arquitecturas con una base experimental previamente comprobada.

4.10. Conclusión del capítulo

Con esta validación, el banco de pruebas queda preparado para abordar topologías de mayor complejidad. La combinación de conectividad, instalación explícita de *flows*, validación del *datapath*, observabilidad del tráfico y microvalidación cuantitativa proporciona una base suficiente para avanzar hacia la evaluación de arquitecturas Leaf-Spine y, posteriormente, Fat-Tree.

A partir de este punto, las siguientes fases del trabajo pueden centrarse en el comportamiento de las topologías y de los perfiles de encaminamiento, apoyándose en un entorno previamente comprobado y trazable. En consecuencia, el capítulo siguiente utiliza la arquitectura Leaf-Spine como etapa metodológica intermedia entre la validación básica del banco de pruebas y el diseño de la topología Fat-Tree multidimensional.

5. LEAF-SPINE COMO ETAPA INTERMEDIA

La topología Leaf-Spine no constituye el objetivo final de este TFG, sino una etapa metodológica intermedia entre las topologías básicas iniciales y la implementación posterior de Fat-Tree. Su incorporación dentro del proyecto responde a una necesidad muy concreta: abandonar los casos simples de conectividad para pasar a un *fabric*, entendido en este contexto como el conjunto de *switches* y enlaces que forman la estructura interna de interconexión de la red, con redundancia real, pero sin introducir todavía la complejidad estructural completa de un Fat-Tree. De este modo, Leaf-Spine se utiliza como banco de pruebas controlado para validar la lógica *multipath* y, sobre todo, el criterio metodológico de instalación, medida e interpretación que más adelante se aplicará sobre una topología Fat-Tree de mayor complejidad.

Dentro de esta etapa no solo se verificó la conectividad extremo a extremo, sino también el comportamiento del plano de datos, la validez del *forwarding* instalado desde el controlador y la forma en que distintas políticas de encaminamiento explotan —o desaprovechan— la redundancia del *fabric*. El capítulo se organiza, por tanto, en dos planos complementarios. En primer lugar, se presenta la topología Leaf-Spine como una arquitectura parametrizable y validable. En segundo lugar, se estudia experimentalmente el efecto de varias políticas de *forwarding* sobre métricas de rendimiento, equidad y utilización de enlaces.

Esta estructura permite que el capítulo no se lea como una colección de pruebas aisladas, sino como una progresión metodológica coherente: primero se construye y valida el banco de pruebas; después se demuestra, con una comparación introductoria, qué significa pasar de camino único a *multipath*; y finalmente se realiza una comparación más madura entre un *baseline* estático bien diseñado y un mecanismo *multipath* general como ECMP. Con ello, Leaf-Spine queda situado exactamente en el papel que le corresponde dentro del TFG: no como destino, sino como puente natural hacia Fat-Tree.

5.1. Topología Leaf-Spine parametrizable

5.1.1. Generación automática

La topología Leaf-Spine utilizada en este trabajo se diseñó de forma parametrizable, de modo que su instancia concreta no dependiera de una construcción manual en Mininet, sino de un conjunto reducido de parámetros estructurales. En particular, la definición de la topología se articula a partir del número de *spines*, el número de *leaves* y el número de *hosts* conectados a cada *leaf*. Esta decisión de diseño tiene un valor metodológico importante, ya que separa claramente la lógica de generación topológica de la ejecución

experimental posterior y facilita tanto la reproducibilidad como la posible ampliación futura del banco de pruebas.

La instancia empleada en este capítulo corresponde a una configuración de 2 *spines*, 4 *leaves* y 2 *hosts* por *leaf*, lo que da lugar a un total de 8 *hosts*. Esta elección no fue arbitraria. Se buscó una topología suficientemente pequeña como para ser fácilmente instrumentable, trazable y defendible, pero al mismo tiempo lo bastante rica como para exhibir de forma clara la diferencia entre un *forwarding* de camino único y un *forwarding multipath*. Con dos *spines* ya existen dos *uplinks* por *leaf*, que es el requisito mínimo para que tenga sentido estudiar reparto de tráfico. Con cuatro *leaves* aparecen suficientes destinos remotos como para construir tráfico concurrente no trivial. Finalmente, con dos *hosts* por *leaf* se consigue un número total de nodos razonable, que permite lanzar varios flujos simultáneos sin disparar la complejidad del análisis.

Dicho de otro modo, la configuración 2-4-2 constituye una instancia pequeña pero no trivial. Es suficientemente simple como para entenderla visualmente y verificarla con detalle, pero suficientemente completa como para representar el fenómeno que se desea estudiar: la existencia de múltiples caminos ascendentes entre origen y destino y la necesidad de decidir cómo utilizarlos. Esta propiedad hace que la topología Leaf-Spine resulte idónea como etapa preparatoria antes de dar el salto a Fat-Tree, donde esa misma filosofía se amplía a una estructura más grande y jerárquica.

La instancia 2-4-2 no se presenta como un dibujo manual aislado, sino como la salida reproducible de un generador parametrizable. La Figura 5.1 permite comprobar visualmente la correspondencia entre los parámetros de diseño y la topología empleada en las pruebas: 2 *spines*, 4 *leaves*, 8 *hosts* y redundancia ascendente suficiente para comparar camino único frente a *multipath*.

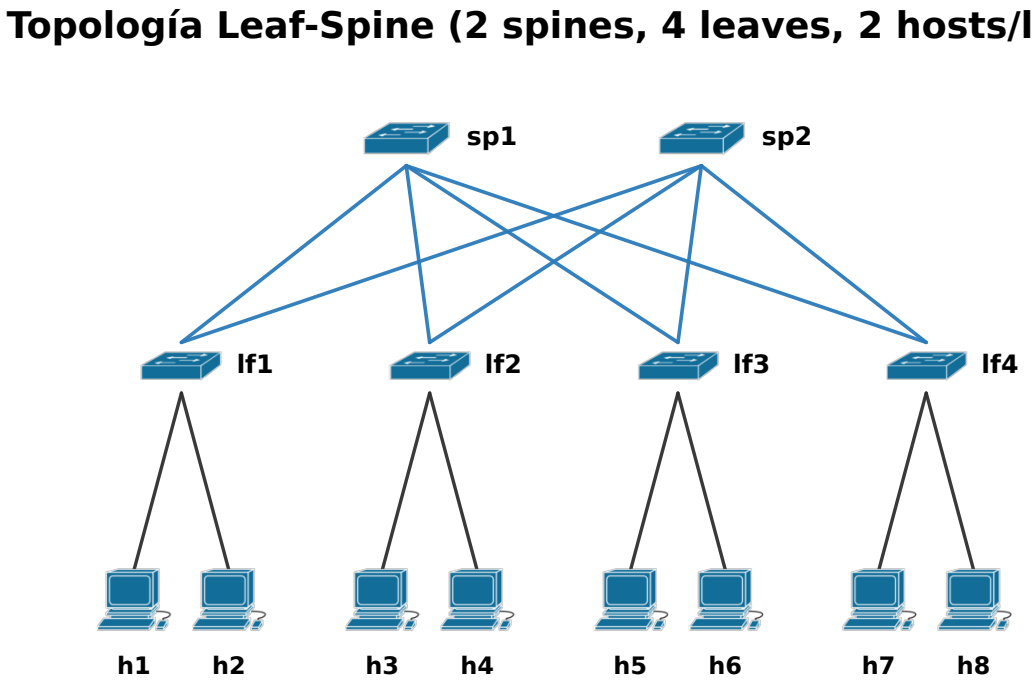

Figura 5.1: Topología Leaf-Spine 2-4-2 generada automáticamente.

5.1.2. Validación de inventario, puertos, *forwarding* y perfiles en OpenDaylight

Una vez generada la topología, el siguiente paso consistió en validar que el banco de pruebas se comportaba de forma controlada antes de iniciar cualquier medición de rendimiento. Esta fase fue importante porque en las primeras pruebas aparecieron problemas asociados al *flooding* ARP y, en general, al comportamiento no deseado del plano de datos cuando la topología dispone de caminos redundantes. Si se hubiese confiado únicamente en el comportamiento por defecto de una red de nivel 2, la existencia de varios caminos habría introducido un tráfico de control difícil de interpretar y poco adecuado para un experimento que pretende medir de manera limpia el efecto de distintas políticas de *forwarding*.

En este contexto podría pensarse en STP como solución clásica para evitar bucles. Sin embargo, en una arquitectura Leaf-Spine eso habría ido en contra del propósito del capítulo. STP resuelve la redundancia bloqueando enlaces, mientras que aquí precisamente se quiere estudiar cómo aprovecharla. Por tanto, la lógica del TFG no consiste en suprimir caminos alternativos, sino en conservarlos y gobernarlos de forma explícita desde SDN. Esa es la razón por la que se optó por una aproximación de *forwarding* controlado en lugar de delegar el comportamiento del *fabric* en mecanismos automáticos de nivel 2.

La validación del banco de pruebas se articuló en varios niveles. En primer lugar, se comprobó desde OpenDaylight el inventario de nodos descubiertos y la correspondencia entre los *switches* Open vSwitch y sus identificadores OpenFlow. En segundo lugar, se verificó en OVS la existencia de *bridges*, puertos, controladoras asociadas, modo seguro

de operación y presencia efectiva de reglas en el *datapath*. En tercer lugar, se validó el *forwarding* instalado para cada perfil experimental, incluyendo flujos IPv4 y, cuando correspondía, estadísticas de grupos OpenFlow. Finalmente, se comprobó la conectividad extremo a extremo con pruebas de `pingall`, asegurando que el banco de pruebas estaba estable antes de ejecutar tráfico concurrente.

En este punto conviene dejar explícito cómo se materializaron los perfiles experimentales desde la controladora. En el perfil *single_path*, el *leaf* de entrada reenvía el tráfico remoto por un único *uplink*, de modo que la topología conserva redundancia física, pero el *datapath* instalado no la explota. En el perfil *L3 static-by-destination*, el reparto se consigue mediante reglas estáticas por destino, asociando manualmente conjuntos de destinos a *uplinks* concretos. En el perfil ECMP, en cambio, se emplea un grupo `select` de OpenFlow para expresar la existencia de varios caminos de igual coste y permitir que el tráfico se distribuya sobre ellos de forma general. Esta explicación conecta la idea teórica de cada perfil con el comportamiento real del *datapath* observado después en las métricas.

El detalle operativo completo de esta configuración incluyendo *payloads* RESTCONF, reglas `add-flow`, definición de grupos OpenFlow `select` y scripts de instalación de *forwarding* se documenta en el Capítulo 3, sección 3.3.2, dedicada al empuje de *flows* y a los perfiles de *forwarding*. De este modo, el presente capítulo puede centrarse en la validación experimental del banco de pruebas, mientras que la descripción detallada de la implementación queda recogida en el bloque técnico donde corresponde.

El *pipeline* experimental del bloque Leaf-Spine se diseñó para mantener una secuencia reproducible desde la generación de la topología hasta la obtención de métricas. La Figura 5.2 representa ese flujo de trabajo: generación, despliegue en Mininet y OVS, descubrimiento por OpenDaylight, validación, aplicación del perfil de *forwarding*, ejecución de tráfico concurrente y procesado de resultados.

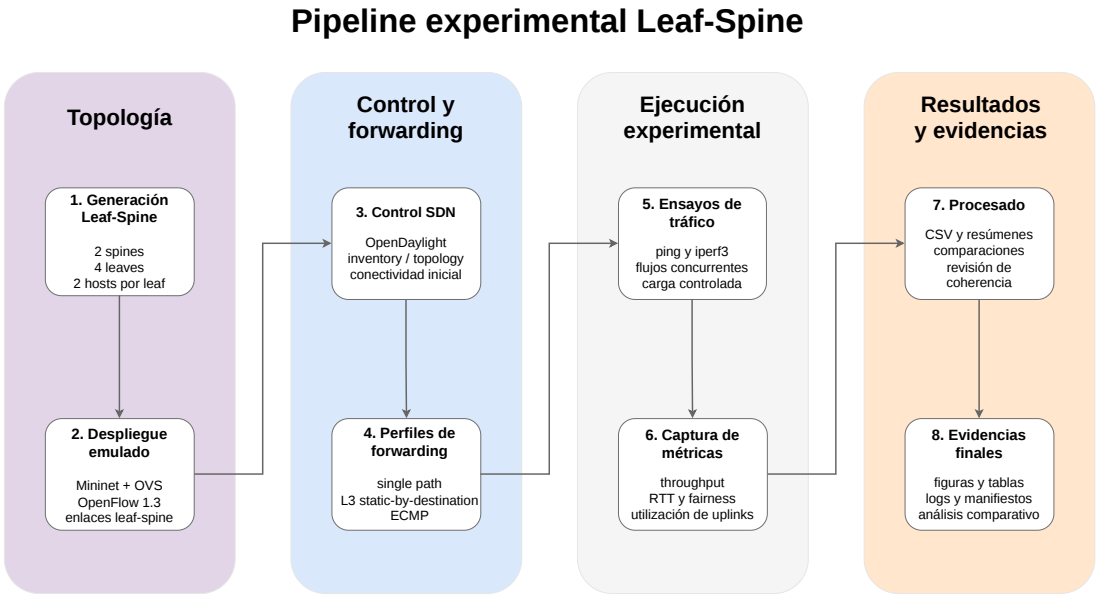

Figura 5.2: Pipeline experimental seguido en el bloque Leaf-Spine.

5.2. Comparación introductoria: *single_path* vs ECMP

5.2.1. Objetivo didáctico de la comparación

La comparación entre *single_path* y ECMP cumple una función eminentemente didáctica dentro del capítulo. Su objetivo no es aún discutir qué política es más elegante o más general, sino mostrar de forma directa y visual qué significa, en una topología Leaf-Spine, pasar de un uso de camino único a un uso real de la redundancia disponible. Es, por tanto, una comparación introductoria, pensada para fijar la intuición principal del lector antes de abordar una comparación más fina entre estrategias que ya explotan los dos *uplinks* del *fabric*.

En el perfil *single_path*, todo el tráfico remoto del *leaf* de entrada se fuerza a salir por un único *uplink*. La conectividad se mantiene, pero uno de los enlaces ascendentes queda convertido en cuello de botella y el otro permanece prácticamente ocioso. En el perfil ECMP, por el contrario, el tráfico remoto puede repartirse entre varios caminos de coste equivalente mediante un grupo `select` de OpenFlow. Esto no cambia el hecho de que la red conecte, pero sí cambia profundamente la forma en que el *fabric* utiliza sus recursos internos.

Desde el punto de vista narrativo, esta comparación es muy útil porque transforma una idea estructural abstracta —"Leaf-Spine tiene redundancia"— en una evidencia experimental fácilmente interpretable. No se trata solo de decir que existen dos *uplinks* por *leaf*, sino de demostrar qué ocurre cuando se decide usar uno solo y qué ocurre cuando se utilizan ambos. La Figura 5.3 fija visualmente esa diferencia conceptual entre camino único y reparto sobre caminos equivalentes.

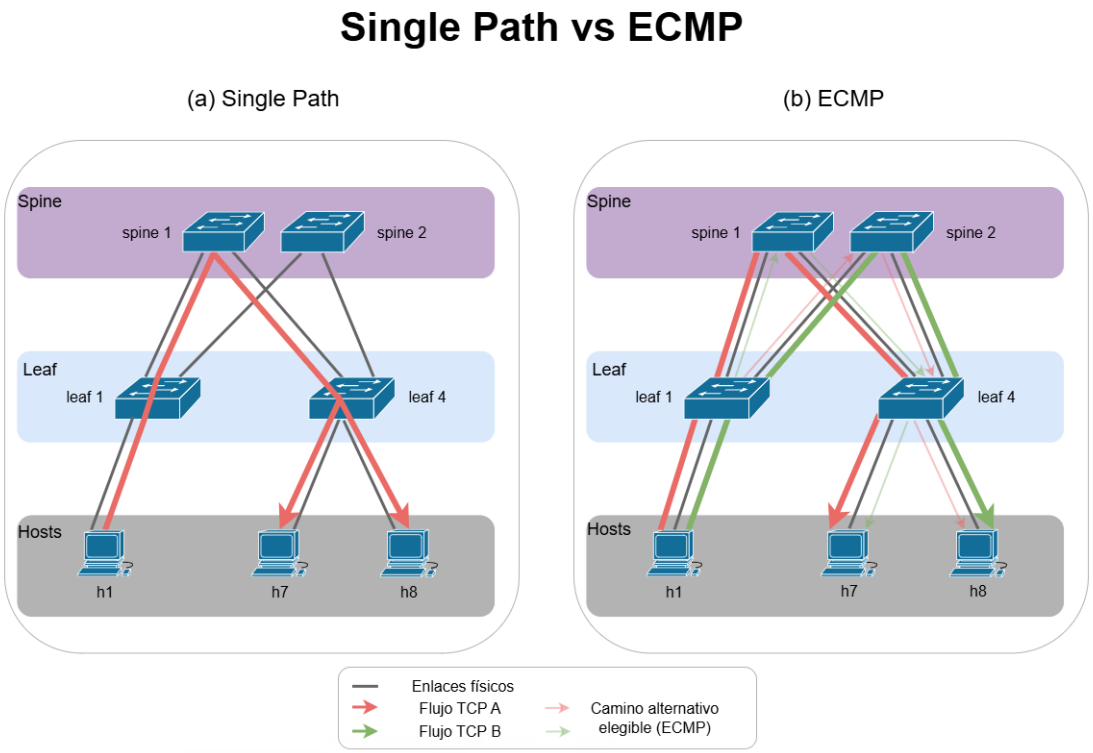

Figura 5.3: Comparación conceptual entre *single_path* y ECMP.

5.2.2. Escenario experimental

El escenario experimental se construyó sobre la instancia Leaf-Spine de 2 *spines*, 4 *leaves* y 2 *hosts* por *leaf* descrita en la sección anterior. Para cada perfil se ejecutaron cinco repeticiones independientes, limitando los enlaces Leaf-Spine a 100 Mbps y manteniendo los enlaces *host-leaf* sin limitación explícita. En cada repetición se lanzaron seis flujos TCP remotos concurrentes y, en paralelo, una medida de latencia mediante `ping` bajo carga. El propósito de esta configuración fue estresar el *leaf* de entrada lo suficiente como para observar con claridad el efecto del mecanismo de *forwarding* sobre el reparto de tráfico y el rendimiento agregado.

Las métricas elegidas fueron el *throughput* agregado, el *throughput* medio por flujo, la *fairness* de Jain, el RTT medio bajo carga y la utilización observada en los dos *uplinks* de `lf1`. El *throughput* agregado representa la suma del rendimiento de todos los flujos concurrentes y, por tanto, es una medida adecuada para evaluar cuánto caudal total consigue transportar el *fabric*. El *throughput* medio por flujo complementa esa visión mostrando cuánto recibe, en promedio, cada flujo individual. La *fairness* de Jain, que toma valores entre 0 y 1, permite cuantificar hasta qué punto el reparto entre flujos es equilibrado. Por último, el RTT bajo carga se incorpora como métrica complementaria para observar si el mecanismo de *forwarding* también afecta al retardo percibido cuando la red está siendo utilizada.

Conviene subrayar que el ancho de banda del enlace no aparece aquí como resultado, sino como condición experimental. Los 100 Mbps constituyen la capacidad configurada de cada enlace Leaf-Spine; las métricas de interés son, en cambio, las que reflejan cómo esa capacidad se aprovecha realmente bajo cada política de *forwarding*.

Esta limitación explícita de los enlaces Leaf-Spine debe tenerse en cuenta al interpretar los resultados. Su finalidad no fue estimar la capacidad absoluta del entorno emulado, sino provocar una saturación controlada que permitiera observar con claridad el efecto de utilizar uno o varios *uplinks*. Por tanto, los valores de *throughput* de este bloque deben compararse únicamente dentro de la propia campaña Leaf-Spine y no de forma directa con los resultados obtenidos posteriormente en Fat-Tree, donde las condiciones experimentales y el objetivo de evaluación son distintos.

5.2.3. Resultados principales

Los resultados muestran una diferencia clara entre ambos perfiles. En *single_path*, el *throughput* agregado medio fue de 95,44 Mbps, con un *throughput* medio por flujo de 15,91 Mbps y una *fairness* de Jain de 0,414. El RTT medio bajo carga alcanzó 18,59 ms. La evidencia más significativa aparece, sin embargo, en la utilización de *uplinks* de `lf1`: el primer *uplink* presentó una utilización media del 96,40 %, mientras que el segundo quedó prácticamente sin uso, con un valor medio de 0,00 %. Esta observación confirma que el perfil *single_path* concentra casi toda la carga ascendente en un único enlace.

Con ECMP, el comportamiento fue claramente distinto. El *throughput* agregado medio ascendió a 191,14 Mbps, aproximadamente el doble del valor observado con *single_path*. El *throughput* medio por flujo se elevó hasta 31,86 Mbps y la *fairness* de Jain mejoró hasta 0,647. Además, el RTT medio bajo carga descendió a 16,31 ms. Desde el punto de vista estructural, el rasgo clave es que los dos *uplinks* de `lf1` quedaron utilizados de forma prácticamente simétrica, con valores medios de 96,22 % y 96,19 %, respectivamente. El *fabric* deja así de comportarse como una arquitectura con redundancia desaprovechada y pasa a explotar de verdad ambos caminos ascendentes.

La lectura académica de estos resultados es sólida. Ambos perfiles conectan correctamente y completan el 100 % de los flujos, de modo que la diferencia entre ellos no es de conectividad, sino de uso interno de la topología. *Single_path* valida que la red funciona, pero lo hace a costa de infrautilizar el *fabric*. ECMP, en cambio, convierte la redundancia estructural de Leaf-Spine en capacidad efectiva.

El *throughput* agregado medio permite visualizar de forma inmediata esa diferencia. La Figura 5.4 muestra que el uso de ECMP prácticamente duplica la capacidad agregada obtenida en el escenario de camino único.

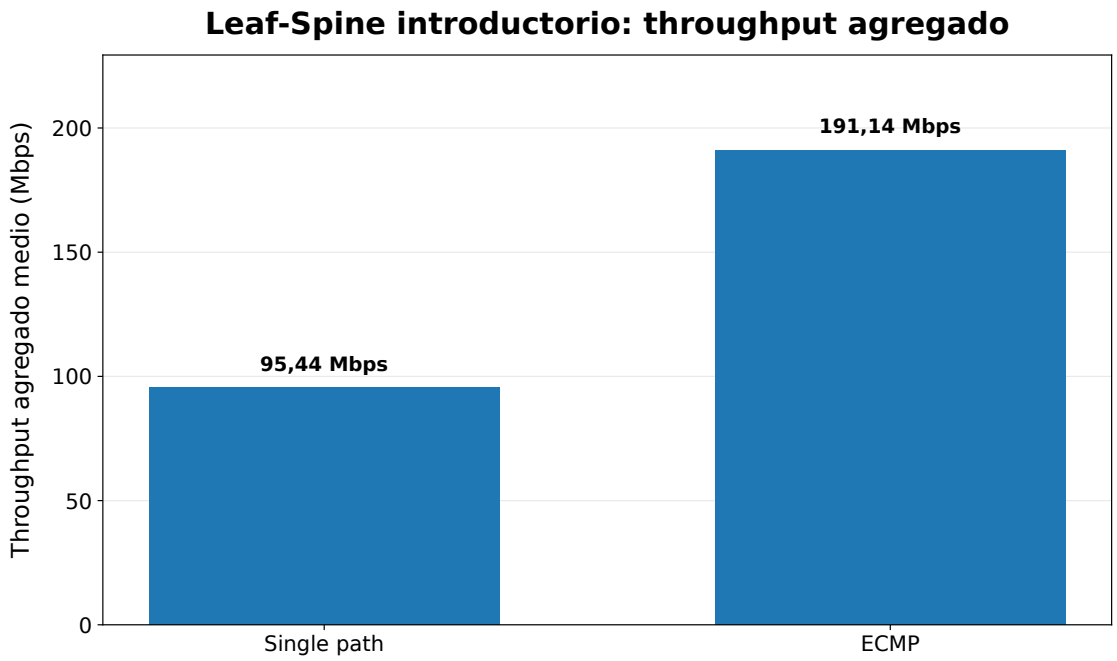

Figura 5.4: Throughput agregado medio: *single_path* vs ECMP.

La utilización de los *uplinks* de `lf1` confirma que la mejora no procede de una variación aleatoria de rendimiento, sino de un cambio estructural en el uso del *fabric*. En *single_path*, la carga se concentra en un único enlace ascendente; con ECMP, ambos enlaces quedan utilizados de forma prácticamente simétrica, como se observa en la Figura 5.5.

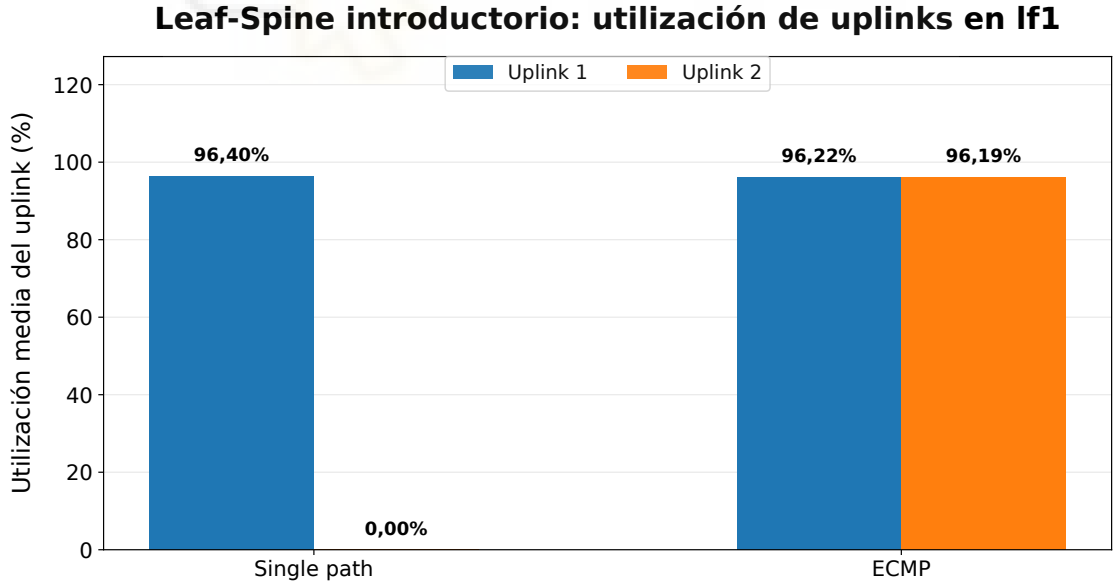

Figura 5.5: Utilización media de los *uplinks* de `lf1`: *single_path* vs ECMP.

La mejora en el uso de enlaces también se refleja en la equidad entre flujos. La Figura 5.6 muestra que el índice de Jain aumenta al pasar de *single_path* a ECMP, lo que indica un reparto relativo más equilibrado cuando se evita concentrar toda la carga remota en un único camino ascendente.

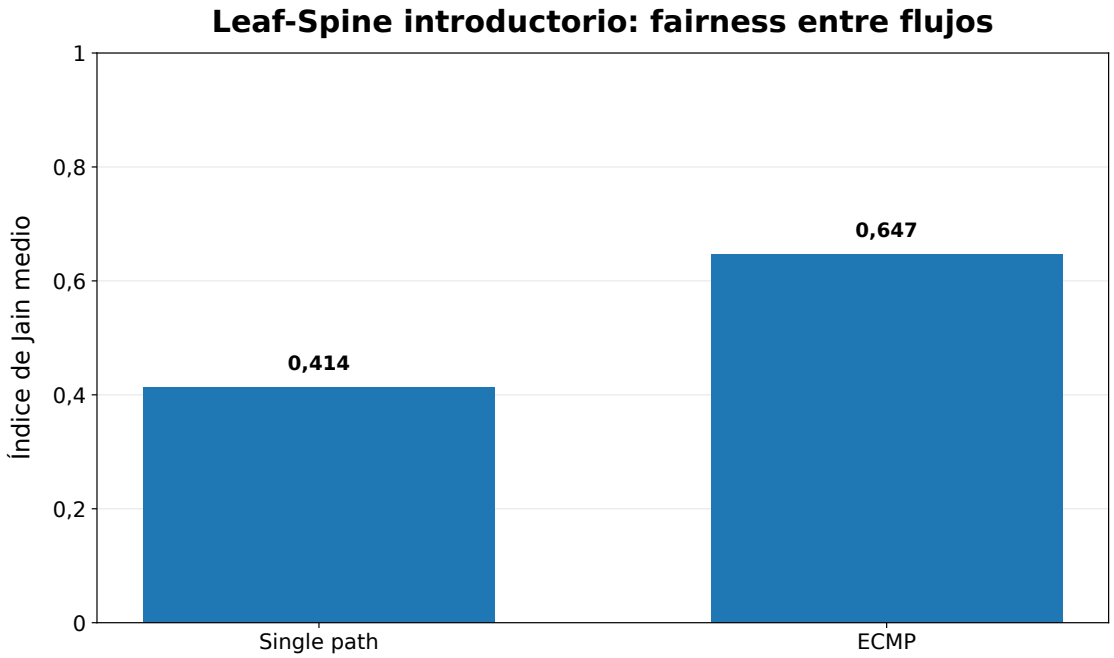

Figura 5.6: Índice de *fairness* de Jain: *single_path* vs ECMP.

El RTT bajo carga no constituye el argumento principal de esta comparación, pero complementa la lectura anterior. La Figura 5.7 muestra que ECMP también presenta un valor medio de latencia más favorable en esta campaña, coherente con la reducción de concentración de tráfico sobre un único enlace.

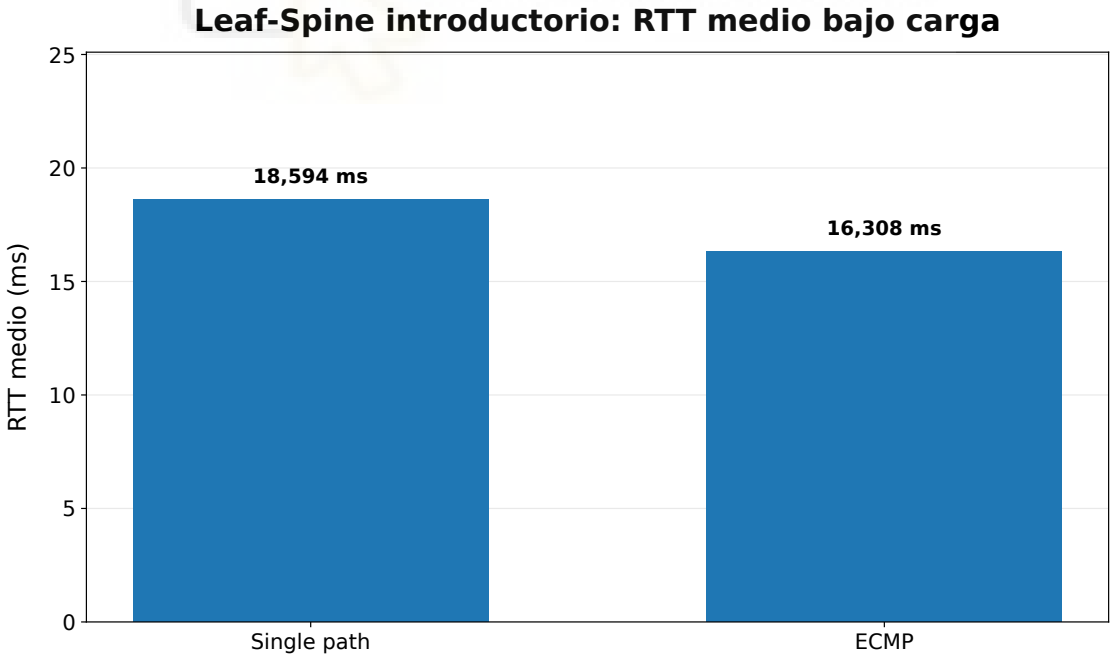

Figura 5.7: RTT medio bajo carga: *single_path* vs ECMP.

La Tabla 5.1 reúne las métricas principales de esta comparación introductoria y permite contrastar en una única vista el efecto de cada perfil sobre rendimiento, equidad, latencia y utilización de *uplinks*.

Tabla 5.1: Resumen cuantitativo de la comparación introductoria Leaf-Spine entre *single_path* y ECMP

Perfil	Flujos OK (%)	Thr. agg. (Mbps)	Thr./flujo (Mbps)	Jain	RTT (ms)	U1 (%)	U2 (%)
single_path	100,00	95,44	15,91	0,414	18,59	96,40	0,00
ECMP	100,00	191,14	31,86	0,647	16,31	96,22	96,19

Nota: U1 y U2 representan la utilización media de los dos *uplinks* de `lf1`. Los valores proceden de cinco repeticiones con enlaces Leaf-Spine limitados a 100 Mbps.

5.2.4. Conclusión parcial

La conclusión de esta comparación introductoria es clara: en una topología Leaf-Spine, la diferencia entre camino único y *multipath* no es cosmética, sino estructural. Cuando el tráfico remoto se fuerza a salir por un único *uplink*, uno de los enlaces ascendentes se convierte en cuello de botella y el rendimiento agregado del *fabric* queda limitado. Cuando el tráfico se reparte entre varios caminos equivalentes, la arquitectura empieza a comportarse de acuerdo con la lógica para la que fue diseñada.

Esta comparación cumple así su función didáctica dentro del capítulo: fijar de forma visual y cuantitativa la idea de que disponer de varios caminos no basta; es necesario contar con un mecanismo de *forwarding* que sepa utilizarlos. Esa intuición prepara el terreno para la comparación siguiente, donde ya no se enfrenta camino único contra *multipath*, sino un *baseline* estático razonable frente a un mecanismo *multipath* general.

5.3. Comparación principal: *L3 static-by-destination* vs ECMP

5.3.1. Justificación del baseline estático

La comparación entre *L3 static-by-destination* y ECMP constituye la comparación principal del capítulo desde un punto de vista metodológico, porque introduce un matiz más maduro que la comparación anterior. Aquí ya no se contrapone un *fabric* mal aprovechado frente a uno bien aprovechado, sino dos estrategias que, en principio, son capaces de utilizar ambos *uplinks* ascendentes del *leaf* de entrada.

El perfil *L3 static-by-destination* consiste en una política de encaminamiento en la que cada destino remoto queda asociado manualmente a un *uplink* concreto. En otras palabras, el reparto se consigue a través de reglas estáticas por destino, no mediante un mecanismo genérico de balanceo. Este *baseline* es defendible porque representa una forma sencilla y comprensible de distribuir tráfico entre varios caminos cuando la topología es pequeña y el conjunto de destinos es manejable. No es una solución absurda ni artificial; al contrario, constituye un punto de comparación útil para evaluar hasta qué punto ECMP aporta ventajas adicionales más allá del rendimiento agregado.

ECMP, por su parte, implementa el reparto *multipath* de forma más general mediante grupos `select` de OpenFlow. La idea esencial es que el *fabric* no necesita decidir manualmente qué destinos salen por qué *uplink*, sino expresar la existencia de varios caminos equivalentes y dejar que el mecanismo distribuya el tráfico sobre ellos. En una topología pequeña ambos enfoques pueden parecer próximos, pero su diferencia conceptual es importante: uno es manual y específico; el otro es general y escalable. La Figura 5.8 separa visualmente la idea de reparto manual por destino de la idea de conjunto de caminos equivalentes elegibles.

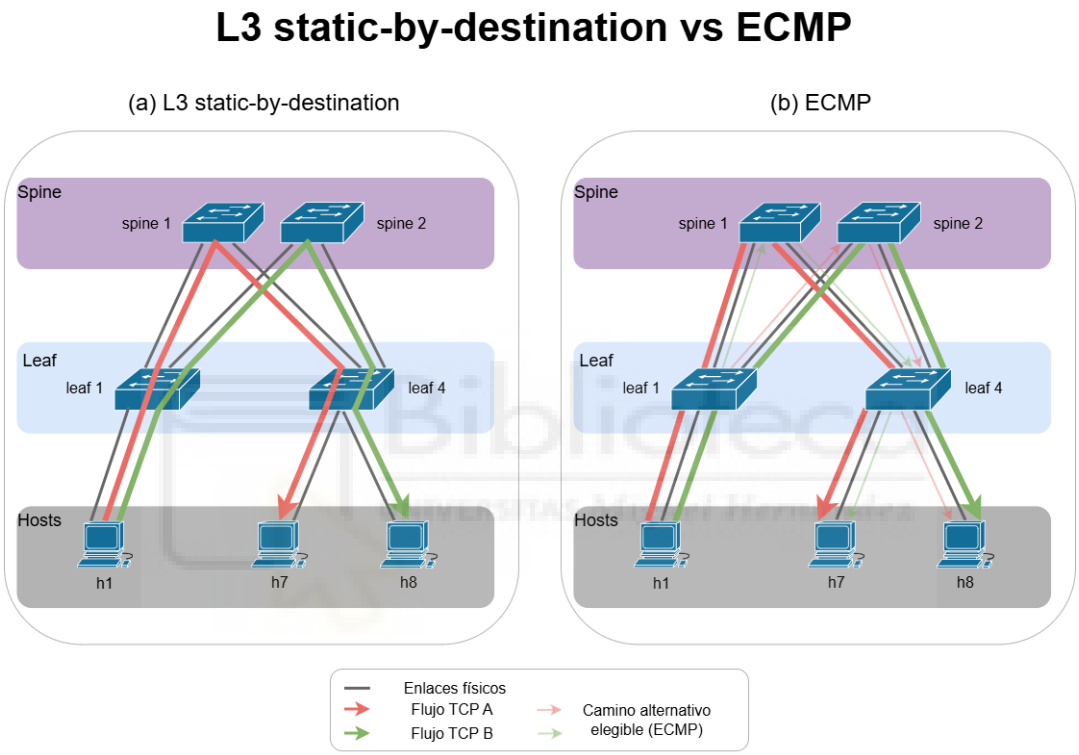

Figura 5.8: Comparación conceptual entre *L3 static-by-destination* y ECMP.

5.3.2. Escenario experimental

Para garantizar comparabilidad, el escenario experimental fue exactamente el mismo utilizado en la comparación introductoria. Se empleó de nuevo la topología Leaf-Spine de 2 *spines*, 4 *leaves* y 2 *hosts* por *leaf*, con seis flujos TCP remotos concurrentes y una medida de RTT mediante `ping` bajo carga. Ambos perfiles se ejecutaron cinco veces y los enlaces Leaf-Spine se limitaron a 100 Mbps.

Las métricas recogidas fueron las mismas que en la comparación anterior: *throughput* agregado, *throughput* medio por flujo, *fairness* de Jain, RTT medio y utilización de los dos *uplinks* de `lf1`. Esta homogeneidad experimental es importante, porque evita que las conclusiones dependan de cambios de escenario y permite atribuir las diferencias observadas exclusivamente a la política de *forwarding* utilizada.

Al igual que en la comparación introductoria, esta campaña debe interpretarse dentro de las condiciones específicas del bloque Leaf-Spine. La limitación de los enlaces Leaf-Spine a 100 Mbps se mantiene como condición experimental para observar saturación controlada y reparto sobre *uplinks*; por tanto, los valores absolutos de *throughput* no se utilizan como comparación directa frente a la campaña Fat-Tree, sino como evidencia interna del comportamiento relativo entre perfiles de *forwarding* en esta topología reducida.

5.3.3. Resultados principales

Los resultados muestran que ambas estrategias presentan un comportamiento agregado muy similar en esta topología concreta. En el caso de *L3 static-by-destination*, el *throughput* agregado medio fue de 191,14 Mbps, con un *throughput* medio por flujo de 31,86 Mbps, una *fairness* de Jain de 0,656 y un RTT medio bajo carga de 17,05 ms. La utilización de los *uplinks* de `lf1` se mantuvo muy equilibrada, con valores medios de 96,12

Con ECMP, el *throughput* agregado medio también se situó en 191,14 Mbps, con un *throughput* medio por flujo de 31,86 Mbps, una *fairness* de Jain de 0,684 y un RTT medio de 14,68 ms. Del mismo modo, ambos *uplinks* de `lf1` presentaron una utilización muy similar, con valores medios de 96,19 % y 96,25 %. Es decir, desde el punto de vista del rendimiento agregado y del reparto de carga en esta topología reducida, ambos perfiles se comportan de manera prácticamente equivalente.

Esta observación es relevante porque evita una lectura simplista del tipo “ECMP siempre mejora mucho cualquier *baseline*”. En este caso, no ocurre así. El *baseline* estático ha sido diseñado con suficiente cuidado como para aprovechar también los dos *uplinks* del *leaf* de entrada y, por tanto, reproducir en gran medida el mismo comportamiento agregado que ECMP. Precisamente por eso esta comparación es metodológicamente más interesante: no enfrenta una mala solución con una buena, sino una solución razonable y específica con una solución general.

El *throughput* agregado medio de ambos perfiles resulta prácticamente idéntico en esta topología reducida. La Figura 5.9 permite visualizar esta equivalencia de rendimiento agregado entre el reparto estático por destino y ECMP.

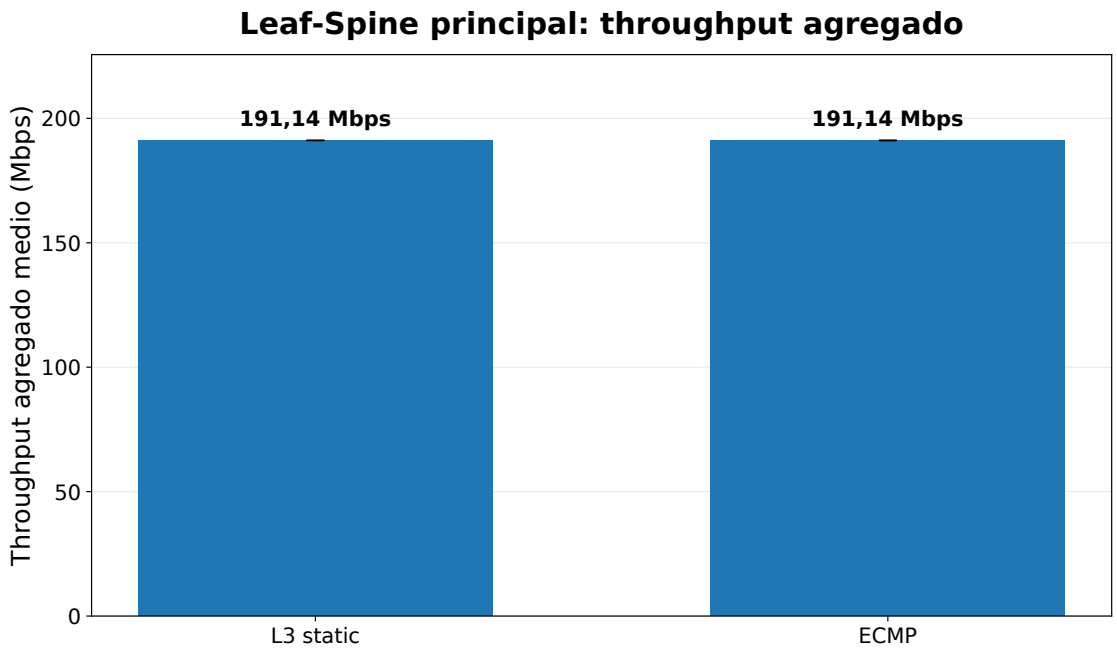

Figura 5.9: Throughput agregado medio: *L3 static-by-destination* vs ECMP.

La equivalencia anterior se explica por el uso equilibrado de los enlaces ascendentes. La Figura 5.10 muestra que ambos perfiles consiguen repartir la carga sobre los dos *uplinks* de `lf1`, aunque el modo de formular ese reparto sea diferente.

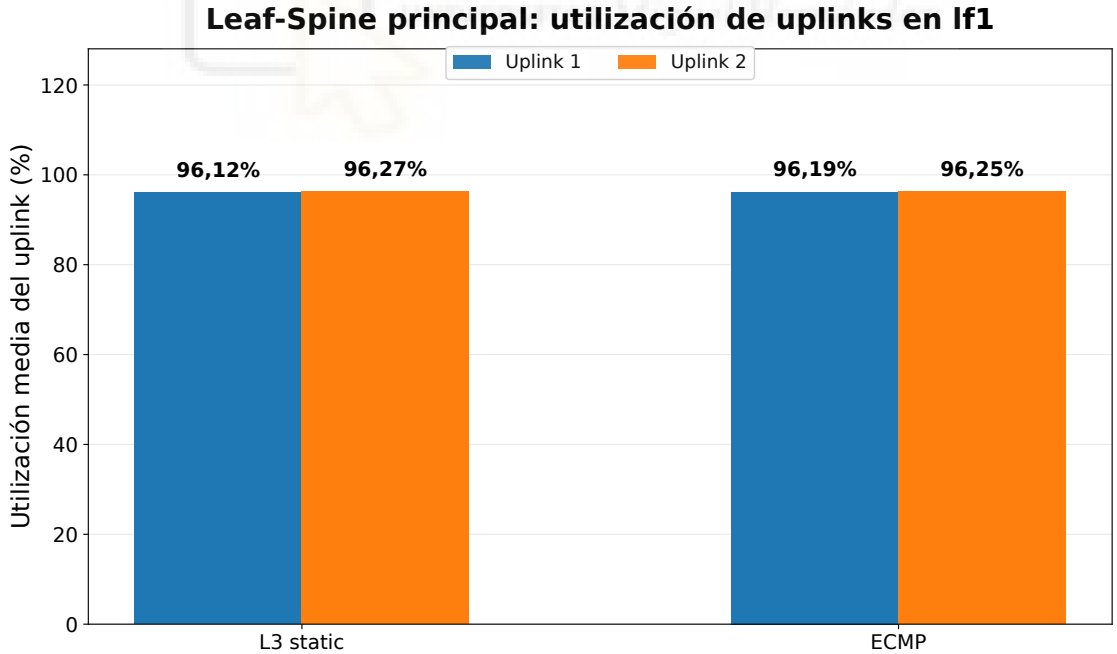

Figura 5.10: Utilización media de los *uplinks* de `lf1`: L3 estático vs ECMP.

La *fairness* de Jain completa esta comparación. La Figura 5.11 muestra una ventaja ligera de ECMP, aunque ambos perfiles presentan valores claramente más equilibrados que los observados en la comparación introductoria con *single_path*.

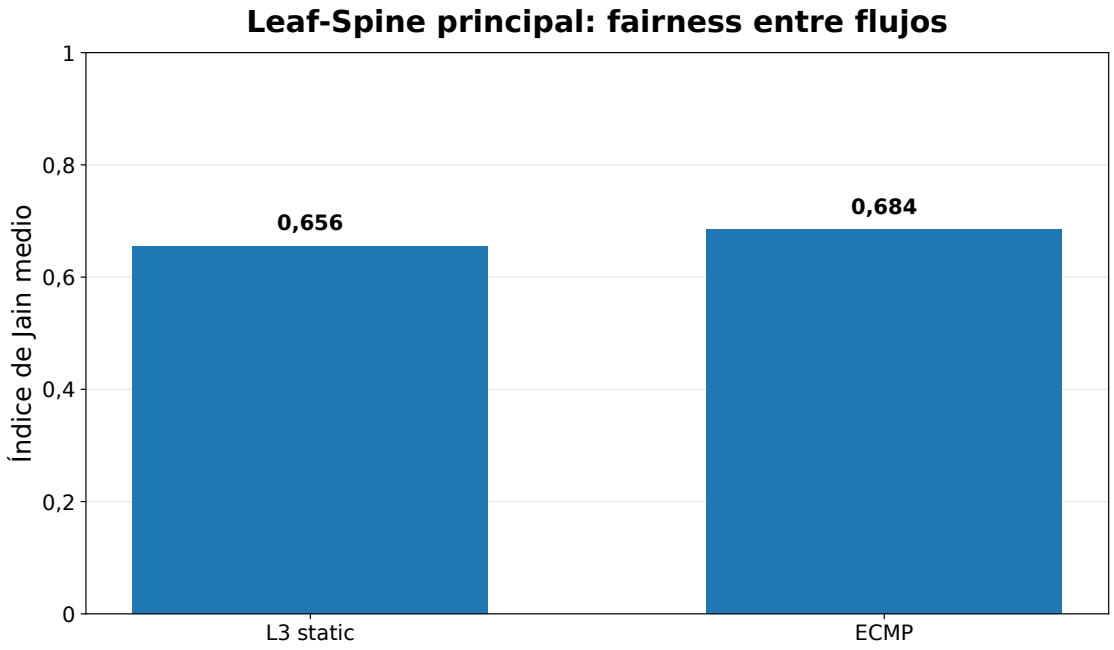

Figura 5.11: Índice de *fairness* de Jain: L3 estático vs ECMP.

La Tabla 5.2 resume cuantitativamente esta comparación principal y permite apreciar que la diferencia fundamental entre ambos perfiles no se encuentra en el rendimiento agregado, sino en la forma de expresar y mantener el reparto *multipath*.

Tabla 5.2: Resumen cuantitativo de la comparación principal Leaf-Spine entre *L3 static-by-destination* y ECMP

Perfil	Flujos OK (%)	Thr. agg. (Mbps)	Thr./flujo (Mbps)	Jain	RTT (ms)	U1 (%)	U2 (%)
L3 estático	100,00	191,14	31,86	0,656	17,05	96,12	96,27
ECMP	100,00	191,14	31,86	0,684	14,68	96,19	96,25

Nota: U1 y U2 representan la utilización media de los dos *uplinks* de `lf1`. Los valores proceden de cinco repeticiones con enlaces Leaf-Spine limitados a 100 Mbps.

5.3.4. Interpretación: reparto manual vs mecanismo general

Aunque las métricas principales sean muy parecidas, ambos mecanismos no deben considerarse equivalentes desde el punto de vista del diseño. En *L3 static-by-destination* el reparto depende de una decisión manual previa: hay que establecer explícitamente qué destinos remotos salen por un *uplink* y cuáles por el otro. Esto puede resultar aceptable en una topología pequeña y perfectamente conocida, pero su mantenimiento se complica a medida que la red crece y aumenta el número de destinos, enlaces y combinaciones posibles.

ECMP expresa esa misma lógica de *multipath* de una forma más general. En lugar de definir el reparto destino a destino, representa directamente la existencia de varios caminos equivalentes y delega la distribución del tráfico en un mecanismo pensado para ese fin. Desde la perspectiva del TFG, esta diferencia es clave. En una topología pequeña, un

baseline estático bien planteado puede aproximar muy bien el rendimiento agregado de ECMP; sin embargo, cuando la topología aumenta en tamaño y complejidad, el coste conceptual y operativo de mantener manualmente ese reparto deja de ser razonable.

Esta conclusión no invalida el uso posterior de *forwarding* determinista como *baseline* experimental en Fat-Tree. En una topología de mayor complejidad, dicho *forwarding* se emplea como mecanismo controlado y trazable para validar la arquitectura, no como propuesta final de balanceo *multipath* escalable. La diferencia es metodológica: una cosa es utilizar rutas deterministas para construir una campaña principal reproducible y otra distinta es defender un reparto manual destino a destino como solución general de explotación de todos los caminos disponibles.

Por tanto, la conclusión de esta comparación principal no es que *L3 static-by-destination* sea inútil, sino precisamente la contraria: es un *baseline* válido y defendible, que ayuda a demostrar que ECMP no se adopta por dogma, sino porque ofrece una formulación más general, menos manual y mejor alineada con *fabrics* escalables. Esa es la idea que enlaza directamente con Fat-Tree.

5.4. Cierre del bloque Leaf-Spine

5.4.1. Lecciones metodológicas

El balance global del capítulo Leaf-Spine deja varias lecciones metodológicas importantes. La primera es que, en topologías con redundancia real, no basta con levantar la conectividad: es necesario estabilizar y validar el comportamiento del plano de datos antes de extraer conclusiones experimentales. La experiencia con el *flooding* ARP y la necesidad de controlar explícitamente el *forwarding* refuerzan esta idea y muestran que una parte esencial del trabajo experimental no consiste solo en medir, sino en garantizar que aquello que se mide tiene una interpretación limpia.

La segunda lección es que la redundancia estructural solo tiene valor si existe un mecanismo capaz de explotarla. La comparación introductoria entre *single_path* y ECMP demuestra que un *fabric* con dos *uplinks* por *leaf* puede comportarse, en la práctica, como si solo tuviera uno si el *forwarding* no está diseñado para repartir tráfico. La tercera lección, más sutil, es que en topologías pequeñas un reparto estático por destino puede aproximar bastante bien el comportamiento agregado de ECMP. Esto obliga a una lectura más honesta y más madura de los resultados: la ventaja de ECMP no reside únicamente en mejorar métricas, sino también en expresar la lógica *multipath* de un modo más general y menos artesanal.

En conjunto, estas observaciones consolidan Leaf-Spine como una etapa metodológica plenamente justificada dentro del proyecto. El capítulo no solo aporta resultados, sino

también criterio experimental y criterio de diseño. No obstante, el alcance de estas conclusiones debe mantenerse acotado al propósito del bloque Leaf-Spine. Los resultados obtenidos validan la capacidad del banco de pruebas para instalar, medir e interpretar mecanismos *multipath* en un entorno reducido y bajo saturación controlada, pero no constituyen una demostración automática de ECMP completo en una topología Fat-Tree. Esa distinción será importante en los capítulos posteriores, donde la mayor complejidad jerárquica exige separar con claridad la campaña principal de validación, las extensiones *multipath* acotadas y los intentos exploratorios hacia ECMP completo.

5.4.2. Por qué Fat-Tree es el siguiente paso lógico

La transición hacia Fat-Tree se justifica de forma natural a partir de las conclusiones obtenidas en Leaf-Spine. Si este capítulo ha permitido demostrar qué significa pasar de camino único a *multipath* y qué diferencia existe entre un reparto manual y un mecanismo general, Fat-Tree constituye el siguiente paso lógico porque amplía precisamente esas dos ideas. En un Fat-Tree aumenta el número de caminos disponibles, crece la estructura jerárquica del *fabric* y se vuelve todavía menos razonable depender de decisiones manuales destino a destino.

Desde esta perspectiva, Leaf-Spine ha cumplido exactamente la función que debía cumplir: servir como banco de pruebas intermedio donde validar la lógica *multipath* en una arquitectura sencilla, visible y defendible. Gracias a ello, el salto a Fat-Tree no se produce como un cambio brusco, sino como una continuación coherente de la misma narrativa experimental. Primero se aprende a construir, validar y medir un *fabric multipath* pequeño; después se traslada esa experiencia metodológica a una topología más rica, más jerárquica y más exigente desde el punto de vista de la interpretación.

En consecuencia, el paso de Leaf-Spine a Fat-Tree no debe interpretarse como una simple ampliación directa del mismo experimento, ni como una garantía de que ECMP completo pueda validarse automáticamente en una arquitectura jerárquica más compleja. Leaf-Spine permite fijar la intuición y validar mecanismos *multipath* en un entorno reducido; Fat-Tree exige una metodología más conservadora, basada primero en *forwarding* determinista trazable y, solo después, en extensiones *multipath* acotadas o exploratorias.

Esa es, en última instancia, la razón por la que Leaf-Spine ocupa este lugar dentro del TFG: no porque sea el punto de llegada, sino porque permite llegar a Fat-Tree con una base metodológica sólida, con una historia experimental ya demostrada y con una lectura prudente sobre los límites de trasladar mecanismos *multipath* desde una topología reducida hacia una arquitectura jerárquica más compleja.

6. DISEÑO E IMPLEMENTACIÓN FAT-TREE

Una vez validado el banco de pruebas SDN y consolidada la topología Leaf-Spine como etapa intermedia, el siguiente paso del proyecto consiste en diseñar e implementar una topología Fat-Tree parametrizable. Este capítulo tiene como objetivo demostrar que dicha arquitectura puede generarse de forma reproducible, instanciarse sobre Mininet y Open vSwitch, ser descubierta por OpenDaylight y validar un conjunto mínimo de caminos representativos en el plano de datos.

A diferencia de los capítulos anteriores, donde se trabajaba con topologías de tamaño reducido para validar progresivamente el entorno, la topología Fat-Tree introduce una estructura jerárquica más rica. En ella aparecen tres niveles principales de conmutación: *switches* de borde, *switches* de agregación y *switches* de núcleo o *core*. Esta organización permite representar de forma más realista una arquitectura de red de centro de datos y sirve como base para los experimentos posteriores de rendimiento, utilización de enlaces y comparación entre variantes.

El propósito de este capítulo no es todavía presentar una evaluación completa de prestaciones. Esa parte queda reservada para los capítulos de metodología experimental y resultados. Aquí se busca cerrar el bloque de diseño e implementación: definir el modelo estructural, justificar la instancia experimental empleada, documentar el generador utilizado, mostrar los metadatos exportados y validar que la topología se integra correctamente en el banco de pruebas SDN.

6.1. Modelo estructural de una Fat-Tree *k*-aria

La topología Fat-Tree utilizada en este trabajo se define a partir de un parámetro par k. Dicho parámetro determina el número de *pods*, el número de *switches* por capa, el número de *hosts* y el número total de enlaces. Esta formulación permite construir topologías de distinto tamaño manteniendo una estructura regular y predecible.

En una Fat-Tree k-aria, la red se divide en k *pods*. Cada *pod* contiene *switches* de borde y *switches* de agregación. Los *hosts* se conectan a los *switches* de borde, mientras que los *switches* de agregación proporcionan conectividad hacia la capa *core*. Esta estructura jerárquica permite separar claramente la conectividad local dentro de un *pod* y la conectividad entre *pods* distintos.

Las expresiones estructurales de la topología permiten dimensionar de forma directa el número de *pods*, *switches*, *hosts* y enlaces a partir del parámetro k. La Tabla 6.1 recoge estas fórmulas y su aplicación a la instancia k=4 utilizada en este proyecto.

Tabla 6.1: Fórmulas estructurales de una topología Fat-Tree k-aria

Magnitud	**Fórmula general**	**k=4**	**Interpretación**
Pods	k	4	Grupos estructurales que contienen *switches* de agregación, *switches* de borde y *hosts*.
Switches *core*	$(k/2)^2$	4	Capa superior de la topología, responsable de interconectar los *pods*.
Switches de agregación	$k \cdot k/2$	8	Capa intermedia dentro de cada *pod*.
Switches de borde	$k \cdot k/2$	8	Capa de acceso a la que se conectan directamente los *hosts*.
Hosts	$k^3/4$	16	Número total de *hosts* emulados en la instancia Fat-Tree.
Enlaces host–edge	$k^3/4$	16	Enlaces de acceso entre *hosts* y *switches* de borde.
Enlaces edge–aggregation	$k^3/4$	16	Enlaces internos de cada *pod* entre *switches* de borde y agregación.
Enlaces aggregation–core	$k^3/4$	16	Enlaces ascendentes desde la capa de agregación hacia la capa *core*.
Enlaces totales	$3k^3/4$	48	Suma de enlaces host–edge, edge–aggregation y aggregation–core.

6.2. Instancia experimental k=4

Para la validación de este capítulo se ha utilizado una instancia Fat-Tree con k=4. Esta elección representa un equilibrio entre riqueza estructural y coste de emulación. La topología resultante incluye cuatro *pods*, cuatro *switches core*, ocho *switches* de agregación, ocho *switches* de borde y dieciséis *hosts*.

La instancia k=4 es suficientemente pequeña para ser ejecutada de forma estable en el entorno Mininet, OVS y OpenDaylight utilizado en este trabajo, pero al mismo tiempo permite validar los elementos esenciales de una Fat-Tree: conectividad local en un *switch* de borde, tránsito dentro de un mismo *pod* y tránsito entre *pods* a través de la capa *core*.

Los parámetros concretos de la instancia experimental se recogen en la Tabla 6.2. Esta tabla fija la escala de trabajo que se utilizará en el resto del capítulo y en las campañas posteriores.

Tabla 6.2: Parámetros de la instancia Fat-Tree k=4

Parámetro	Valor	Justificación
Valor de k	4	Instancia canónica suficientemente rica para representar una arquitectura Fat-Tree, pero manejable en un entorno emulado local.
Pods	4	Derivado directamente del valor de k.
Switches *core*	4	Capa superior de interconexión entre *pods*.
Switches de agregación	8	Dos *switches* de agregación por *pod*.
Switches de borde	8	Dos *switches* de borde por *pod*.
Hosts	16	Cuatro *hosts* por *pod*, suficientes para validar conectividad multi-*pod* y generación de tráfico entre ramas distintas.
Switches totales	20	Suma de *switches core*, agregación y borde.
Enlaces totales	48	Incluye enlaces host–edge, edge–aggregation y aggregation–core.
Protocolo OpenFlow	OpenFlow 1.3	Coherente con el banco de pruebas validado en capítulos anteriores.
Controladora	OpenDaylight Titanium	Controladora SDN utilizada en el entorno experimental.
Datapath	Open vSwitch	*Switch* virtual empleado por Mininet para la emulación del plano de datos.

Como se observa a partir de su dimensionamiento, la topología contiene veinte *switches* y cuarenta y ocho enlaces. Esta escala resulta adecuada para validar el diseño sin introducir una carga excesiva en el controlador ni en el entorno de emulación.

6.3. Implementación parametrizada en Python sobre Mininet

La implementación de la topología se ha realizado mediante un generador Python integrado con Mininet. En lugar de definir la red manualmente, el generador construye la especificación de la topología a partir del parámetro k. Esto permite mantener una relación directa entre el modelo teórico y la topología realmente instanciada.

El generador se encarga de crear los *switches* de las tres capas principales, asignar nombres deterministas, crear los *hosts*, definir direcciones IP y MAC, y construir los enlaces con puertos explícitos. Esta última decisión es especialmente importante, ya que permite razonar posteriormente sobre los caminos de *forwarding* y sobre las reglas OpenFlow instaladas.

El extracto del Código 6.1 representa la parte inicial del generador, donde se valida el parámetro k, se inicializan las estructuras de metadatos y se construye de forma determinista la capa *core*.

```python
def build_fattree_spec(k: int = 4) -> dict[str, Any]:
    if k < 4 or k % 2 != 0:
        raise ValueError("Fat-Tree requiere un valor par de k y k >= 4.")

    half = k // 2
    nodes = []
    links = []
    hostmap = []

    # Core switches: c1..cN
    for group in range(half):
        for pos in range(half):
            idx = group * half + pos + 1
            nodes.append({
                "name": f"c{idx}",
                "type": "switch",
                "layer": "core",
                "index": idx,
            })

    # Pods: aggregation, edge and hosts
    host_id = 1
    for pod in range(k):
        # aggregation and edge switches are created per pod
        # hosts are connected deterministically to edge switches
        # links are exported with src_port and dst_port metadata
        ...
```

Código 6.1: Generador parametrizable de la topología Fat-Tree.

Además de crear la topología para Mininet, el generador exporta metadatos en formato JSON y CSV. Estos metadatos se utilizan como evidencia documental y como fuente común para construir tablas, figuras y comprobaciones posteriores. El Código 6.2 recoge la ejecución de exportación y los archivos generados por el proceso.

```
$ python3 tfg_sdn/mininet/topos/fattree.py --k 4 --export --outdir data/ch06_fat_tree

DONE: Fat-Tree metadata exported.

MANIFEST: data/ch06_fat_tree/ch06_fattree_k4_manifest.json
NODES:    data/ch06_fat_tree/ch06_fattree_k4_nodes.csv
LINKS:    data/ch06_fat_tree/ch06_fattree_k4_links.csv
HOSTMAP:  data/ch06_fat_tree/ch06_fattree_k4_hostmap.csv
PORTS:    data/ch06_fat_tree/ch06_fattree_k4_ports.csv
```

Código 6.2: Exportación de metadatos de la topología Fat-Tree k=4.

La exportación de metadatos es una parte importante de la metodología seguida en este trabajo. Permite mantener trazabilidad entre código, documentación y evidencias. En particular, los archivos exportados incluyen el manifiesto global de la topología, la lista de nodos, la lista de enlaces, el mapeo de *hosts* y la política de puertos.

6.4. Metadatos, direccionamiento y política de puertos

La instancia Fat-Tree k=4 se documenta mediante un conjunto de metadatos generados automáticamente. Estos archivos permiten verificar que la topología contiene los elementos esperados y que la asignación de *hosts*, *switches* y enlaces es coherente con el modelo estructural.

El direccionamiento empleado sigue una política sencilla y determinista. Los *hosts* se nombran como `h1`, `h2`, . . . , `h16`, y reciben direcciones IP dentro del rango `10.0.0.X/24`. Los *switches* se nombran según su capa: `c` para la capa *core*, `a` para agregación y `e` para borde.

La distribución de *hosts* por *pod* permite comprobar que la instancia mantiene una estructura regular: cada *pod* contiene dos *switches* de borde y cuatro *hosts*. La Tabla 6.3 recoge este mapeo resumido.

Tabla 6.3: Mapeo resumido de hosts en Fat-Tree k=4

Pod	Switches de borde	Hosts	Rango IP	Observación
1	`e1`, `e2`	`h1`–`h4`	`10.0.0.1`–`10.0.0.4`	Primer *pod* de la topología, formado por dos *switches* de borde y cuatro *hosts*.
2	`e3`, `e4`	`h5`–`h8`	`10.0.0.5`–`10.0.0.8`	Segundo *pod*, con la misma estructura interna que el resto de *pods*.
3	`e5`, `e6`	`h9`–`h12`	`10.0.0.9`–`10.0.0.12`	Tercer *pod*, utilizado para validar conectividad entre ramas distintas.
4	`e7`, `e8`	`h13`–`h16`	`10.0.0.13`–`10.0.0.16`	Cuarto *pod* de la instancia Fat-Tree k=4.

La política de puertos también se define de forma determinista. En los *switches* de borde, los primeros puertos se reservan para *hosts* y los siguientes para enlaces hacia agregación. En los *switches* de agregación, los primeros puertos conectan con *switches* de borde y los siguientes con la capa *core*. La Tabla 6.4 recoge esta política por tipo de nodo.

Tabla 6.4: Política determinista de puertos en Fat-Tree k=4

Tipo de nodo	Puertos inferiores	Puertos superiores	Uso en la topología
Switch *core*	Puertos 1–4 hacia *pods*	No aplica	Interconecta los *pods* de la topología mediante enlaces hacia *switches* de agregación.
Switch de agregación	Puertos 1–2 hacia borde	Puertos 3–4 hacia *core*	Actúa como capa intermedia dentro de cada *pod* y proporciona conectividad hacia la capa *core*.
Switch de borde	Puertos 1–2 hacia *hosts*	Puertos 3–4 hacia agregación	Conecta directamente los *hosts* del *pod* y proporciona enlaces ascendentes hacia la capa de agregación.
Host	No aplica	Puerto 1 hacia borde	Nodo terminal emulado por Mininet, con dirección IP determinista `10.0.0.X/24`.

Esta política de puertos permite razonar de manera explícita sobre rutas representativas. Por ejemplo, un camino entre *hosts* conectados al mismo *switch* de borde no necesita atravesar la capa de agregación, mientras que un camino entre *pods* distintos debe ascender desde borde a agregación, pasar por *core* y descender de nuevo hacia el *pod* de destino.

6.5. Representación visual de la topología

La representación visual de la instancia Fat-Tree k=4 se ha generado a partir de los metadatos exportados por el propio generador, por lo que mantiene coherencia con los archivos CSV y JSON utilizados en el resto del capítulo. La Figura 6.1 permite identificar las capas *core*, agregación, borde y *hosts*, así como la división lógica en cuatro *pods*.

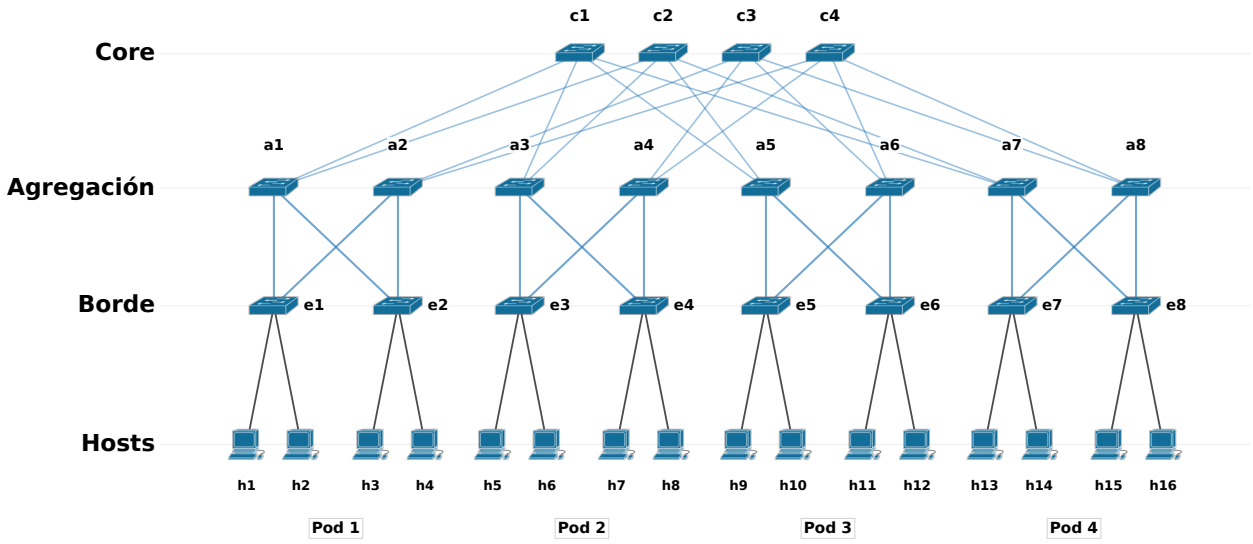

Figura 6.1: Topología Fat-Tree k=4 generada a partir de los metadatos exportados.

La figura muestra que los *hosts* se conectan a los *switches* de borde; estos se conectan a los *switches* de agregación dentro de cada *pod*; y la capa de agregación proporciona acceso a los *switches core*. Esta estructura será la base sobre la que se plantearán las pruebas de rendimiento y los escenarios de tráfico posteriores.

6.6. Validación de instanciación SDN

Una vez definida la topología y generados sus metadatos, se validó su instanciación dentro del banco de pruebas SDN. Esta validación comprueba que Mininet crea correctamente la topología, que Open vSwitch genera los *bridges* esperados, que los *switches* se conectan al controlador y que OpenDaylight descubre los nodos OpenFlow correspondientes.

La validación de instanciación combina evidencias del plano de datos y del plano de control. En OVS se comprueba la presencia de los *bridges*, la conexión con la controladora, el uso de OpenFlow 1.3 y el modo seguro. En OpenDaylight se comprueba el inventario RESTCONF y la topología operacional descubierta. La Tabla 6.5 resume esta comprobación para la instancia k=4.

Tabla 6.5: Validación de instanciación SDN de Fat-Tree k=4

Elemento validado	**Resultado**	**Evidencia generada**
Instanciación de switches OVS	20/20 *bridges* Fat-Tree	Archivo `ch06_fattree_k4_ovs_bridges.txt` con *bridges* `c1-c4`, `a1-a8` y `e1-e8`.
Conexión al controlador SDN	20/20 conectados	Archivo `ch06_fattree_k4_controller_connection_state.txt` con `is_connected: true` y estado `ACTIVE`.
Configuración OpenFlow	OpenFlow 1.3 y modo seguro	Archivo `ch06_fattree_k4_ovs_controller_protocols.txt` con `protocols: [OpenFlow13]` y `fail_mode: secure`.
Inventario RESTCONF	20/20 nodos OpenFlow	Archivo `ch06_fattree_k4_inventory_nodes.json` con nodos `openflow:1` a `openflow:20`.
Topología RESTCONF	20/20 nodos OpenFlow	Archivo `ch06_fattree_k4_network_topology.json` con nodos `openflow:1` a `openflow:20`.
Evolución temporal del descubrimiento	`t=60s inventory=20 topology=20`	Archivo `ch06_fattree_k4_odl_discovery_progress.txt` con seis muestras consecutivas.

El resultado confirma que se detectaron veinte *bridges* OVS asociados a la topología Fat-Tree y que todos ellos quedaron configurados hacia el controlador en `tcp:127.0.0.1:6653`. También se comprobó el uso de OpenFlow 1.3, el modo `secure` y el estado activo de las conexiones hacia OpenDaylight.

El Código 6.3 conserva el extracto de validación en tiempo de ejecución, incluyendo la evolución temporal del descubrimiento y el estado final de conexión de los *bridges* OVS.

```
Validacion SDN en tiempo de ejecucion para Fat-Tree k=4

ODL discovery progress:
t=10s inventory=20 topology=20
t=20s inventory=20 topology=20
t=30s inventory=20 topology=20
t=40s inventory=20 topology=20
t=50s inventory=20 topology=20
t=60s inventory=20 topology=20

RESTCONF summary:
inventory_openflow_nodes: 20
inventory_expected_1_20_present: True
inventory_missing: none

topology_openflow_nodes: 20
topology_expected_1_20_present: True
topology_missing: none

OVS-controller state:
ovs_fattree_bridges: 20/20
ovs_controller_target: tcp:127.0.0.1:6653
ovs_connected_controllers: 20/20
ovs_active_controllers: 20/20
ovs_fail_mode: secure
ovs_protocol: OpenFlow13
```

Código 6.3: Validación SDN en tiempo de ejecución de la topología Fat-Tree k=4.

Las consultas RESTCONF posteriores confirmaron la presencia de los veinte nodos OpenFlow esperados tanto en el inventario operativo como en la topología descubierta por OpenDaylight. Por tanto, esta evidencia permite afirmar que la topología no solo se genera en Mininet, sino que también queda integrada en el plano de control SDN.

Esta validación debe interpretarse como una comprobación de instanciación y descubrimiento. No pretende demostrar por sí sola conectividad extremo a extremo entre todos los *hosts*, sino confirmar que la infraestructura necesaria para dicha conectividad está correctamente desplegada.

6.7. Validación funcional mínima de *forwarding*

Tras validar la instanciación, se realizó una validación funcional mínima del plano de datos. Para ello se seleccionaron tres casos representativos que cubren distintos niveles de complejidad dentro de la topología:

- comunicación entre *hosts* conectados al mismo *switch* de borde;
- comunicación entre *hosts* situados en el mismo *pod*, pero en *switches* de borde distintos;
- comunicación entre *hosts* ubicados en *pods* distintos, atravesando la capa *core*.

Los tres casos permiten comprobar caminos locales, intra-*pod* e inter-*pod*. La Tabla 6.6 resume los resultados de conectividad, pérdida y latencia obtenidos en esta validación mínima.

Tabla 6.6: Validación funcional mínima de *forwarding* en Fat-Tree k=4

Caso	Ámbito	Ruta lógica validada	Rx/Tx	Pérdida	RTT medio	Interpretación
`h1→h2`	Mismo switch de borde	`e1`	5/5	0 %	0,204 ms	Valida *forwarding* local en el nivel de borde.
`h1→h3`	Mismo pod	`e1→a1→e2`	5/5	0 %	0,233 ms	Valida tránsito edge–aggregation dentro de un *pod*.
`h1→h16`	Pods distintos	`e1→a1→c1 →a7→e8`	5/5	0 %	0,300 ms	Valida tránsito completo entre *pods* a través del *core*.

Nota: Las consultas RESTCONF posteriores a la prueba devolvieron `HTTP=200` y confirmaron la presencia de 20 nodos OpenFlow tanto en *inventory* como en *network-topology*. Las reglas se instalaron de forma determinista en OVS para validar el plano de datos de la instancia Fat-Tree.

Los tres casos presentan 0 % de pérdida de paquetes. El primer caso valida *forwarding* local en el nivel de borde. El segundo caso valida tránsito dentro de un mismo *pod* a través de la capa de agregación. El tercer caso valida un camino inter-*pod* que asciende hasta la capa *core* y desciende hacia el *pod* de destino.

Además de los resultados de conectividad, se conservaron contadores OpenFlow y comprobaciones RESTCONF posteriores a la prueba. El Código 6.4 recoge esa evidencia de bajo nivel, mostrando que las reglas asociadas a los caminos representativos recibieron tráfico.

```
Validacion funcional de forwarding para Fat-Tree k=4

Representative pings:
h1_h2_same_edge        rx/tx=5/5 loss=0% rtt_avg=0.204 ms
h1_h3_same_pod         rx/tx=5/5 loss=0% rtt_avg=0.233 ms
h1_h16_cross_pod       rx/tx=5/5 loss=0% rtt_avg=0.300 ms

Observed OpenFlow counters on relevant switches:
e1 dst=10.0.0.1   packets=14 output:1
e1 dst=10.0.0.2   packets= 5 output:2
e1 dst=10.0.0.3   packets= 5 output:3
e1 dst=10.0.0.16  packets= 4 output:3
e2 dst=10.0.0.1   packets= 5 output:3
e2 dst=10.0.0.3   packets= 5 output:1
a1 dst=10.0.0.1   packets= 9 output:1
a1 dst=10.0.0.3   packets= 5 output:2
a1 dst=10.0.0.16  packets= 4 output:3
c1 dst=10.0.0.1   packets= 4 output:1
c1 dst=10.0.0.16  packets= 4 output:4
a7 dst=10.0.0.1   packets= 4 output:3
a7 dst=10.0.0.16  packets= 4 output:2
e8 dst=10.0.0.1   packets= 4 output:3
e8 dst=10.0.0.16  packets= 4 output:2

RESTCONF:
inventory_http=200
topology_http=200
inventory_openflow_nodes=20/20
topology_openflow_nodes=20/20

Note:
table-miss drop counters are not used as forwarding proof.
They only indicate discarded traffic outside the explicit IP rules.
```

Código 6.4: Validación funcional mínima de *forwarding* en la topología Fat-Tree k=4.

Los contadores OpenFlow permiten comprobar que las reglas asociadas a los caminos representativos recibieron tráfico. Esta evidencia no debe interpretarse como una validación exhaustiva de todos los pares de *hosts*, sino como una comprobación funcional mínima del plano de datos sobre caminos representativos de complejidad creciente.

Asimismo, conviene remarcar que en esta fase las reglas de *forwarding* se instalaron de forma determinista para validar el comportamiento de la topología y del plano de datos. No se afirma que OpenDaylight haya calculado automáticamente rutas *multipath* para la topología Fat-Tree. La validación se centra en demostrar que la instancia k=4 puede integrarse con el controlador y transportar tráfico correctamente en caminos seleccionados.

6.8. Escalabilidad teórica del diseño

Aunque la instancia experimental utilizada es k=4, el modelo implementado es parametrizable. Para mostrar esta propiedad, se generó una tabla de escalabilidad teórica para distintos valores de k. Esta tabla permite observar cómo crece el número de *switches*, *hosts* y enlaces al aumentar el tamaño de la topología.

La escalabilidad teórica se ha calculado para k=4, k=6 y k=8. La Tabla 6.7 reúne estos valores y permite comparar el crecimiento estructural de la topología.

Tabla 6.7: Escalabilidad teórica de Fat-Tree según el parámetro k

k	Pods	Core	Agregación	Borde	Hosts	Switches	Enlaces
4	4	4	8	8	16	20	48
6	6	9	18	18	54	45	162
8	8	16	32	32	128	80	384

Nota: La tabla se obtiene directamente de las expresiones estructurales de una Fat-Tree k-aria: k *pods*, $(k/2)^2$ *switches core*, $k^2/2$ *switches* de agregación, $k^2/2$ *switches* de borde, $k^3/4$ *hosts* y $3k^3/4$ enlaces. En el banco de pruebas se emplea k=4 como instancia experimental manejable dentro del entorno emulado.

El crecimiento del número de *hosts* y enlaces es rápido, especialmente a partir de valores superiores de k. Por este motivo, la elección de k=4 resulta razonable para la fase experimental inicial del proyecto. Permite validar la arquitectura completa sin comprometer la estabilidad del banco de pruebas.

Los valores k=6 y k=8 no se presentan como instancias levantadas experimentalmente en este capítulo, sino como una proyección teórica derivada de las fórmulas estructurales. Su inclusión permite mostrar que la implementación no se limita conceptualmente a la instancia k=4, aunque la experimentación práctica se centre en una escala manejable.

6.9. Conclusiones del capítulo

En este capítulo se ha diseñado e implementado una topología Fat-Tree (k)-aria como evolución natural del banco de pruebas validado en los capítulos anteriores. La implementación se ha realizado mediante un generador Python parametrizable, capaz de construir la topología, asignar nombres y direcciones, definir enlaces con puertos explícitos y exportar metadatos reproducibles.

La instancia experimental (k=4) queda documentada mediante tablas, figura, listados y archivos auxiliares. Esta instancia contiene veinte *switches*, dieciséis *hosts* y cuarenta y ocho enlaces, lo que permite representar una arquitectura jerárquica completa con capas de borde, agregación y *core*.

La validación realizada confirma la instanciación correcta en Mininet/OVS, la conexión de los *bridges* al controlador mediante OpenFlow 1.3 y el descubrimiento de los veinte nodos OpenFlow esperados. Además, se ha validado *forwarding* funcional mínimo en tres caminos representativos: local, intra-*pod* e inter-*pod* atravesando la capa *core*.

Con ello, el diseño e implementación Fat-Tree queda cerrado. Esta topología servirá de base para la metodología experimental posterior y para analizar rendimiento, utilización, latencia, reparto de carga y comportamiento ante distintos escenarios de tráfico.

7. METODOLOGÍA EXPERIMENTAL

Una vez diseñada, implementada y validada funcionalmente la topología Fat-Tree k=4, el siguiente paso consiste en definir una metodología experimental que permita evaluarla de forma reproducible dentro del banco de pruebas SDN desarrollado. El objetivo de este capítulo no es presentar todavía resultados cuantitativos, sino establecer el marco de evaluación que se utilizará posteriormente: matriz experimental, escenarios de tráfico, métricas previstas, procedimiento de ejecución, tratamiento de artefactos y limitaciones metodológicas.

La metodología se formula de forma conservadora, teniendo en cuenta que el entorno empleado se basa en Mininet, Open vSwitch y OpenDaylight sobre una plataforma local de ejecución emulada. Por tanto, las medidas obtenidas deberán interpretarse como resultados internos del banco de pruebas y no como prestaciones absolutas extrapolables directamente a una infraestructura física de centro de datos. Esta distinción resulta especialmente importante para mantener una interpretación rigurosa de los resultados que se presentarán en el capítulo siguiente y para evitar comparaciones directas entre campañas que responden a objetivos experimentales distintos.

7.1. Objetivo y alcance de la evaluación

El objetivo de la evaluación experimental es caracterizar el comportamiento de la instancia Fat-Tree k=4 dentro del banco de pruebas SDN implementado. La evaluación se plantea sobre una topología ya generada de forma parametrizada, con metadatos exportados, direccionamiento definido, política de puertos documentada y validación funcional mínima previa. De este modo, el Capítulo 7 parte de una base técnica ya consolidada en el Capítulo 6 y se centra en definir cómo deben ejecutarse y tratarse los ensayos experimentales.

El alcance de la evaluación se limita al entorno emulado compuesto por Mininet, Open vSwitch y OpenDaylight. Esta decisión permite trabajar en un banco de pruebas controlado, inspeccionable y reproducible, pero también obliga a interpretar las métricas con cautela. Los valores absolutos de *throughput*, RTT o utilización no deben entenderse como equivalentes a los de una red física de producción, sino como medidas obtenidas bajo unas condiciones concretas de emulación y configuración.

La metodología se estructura en torno a cinco elementos principales. En primer lugar, se define una matriz experimental que recoge los tipos de ensayo previstos. En segundo lugar, se concretan escenarios de tráfico representativos de distinta complejidad topológica. En tercer lugar, se clasifican las métricas previstas según su grado de consolidación. En cuarto lugar, se establece un procedimiento experimental reproducible. Finalmente, se

define cómo se tratarán los artefactos generados y qué limitaciones deben tenerse en cuenta durante la interpretación.

Esta separación evita mezclar metodología y resultados. El presente capítulo define el marco de trabajo; los valores medidos, las gráficas comparativas y la interpretación cuantitativa se reservan para el Capítulo 8.

7.2. Matriz experimental y escenarios de tráfico

La evaluación se organiza mediante una matriz experimental que separa ensayos de conectividad representativa, pruebas de rendimiento punto a punto y escenarios con tráfico concurrente. Esta separación permite avanzar desde comunicaciones sencillas, utilizadas como referencia mínima, hasta patrones de tráfico más exigentes dentro de la topología Fat-Tree.

La matriz experimental no debe interpretarse como una tabla de resultados, sino como una planificación metodológica. Su objetivo es delimitar qué se evaluará y qué función cumple cada tipo de ensayo dentro de la campaña. La Tabla 7.1 recoge esta planificación.

Tabla 7.1: Matriz experimental del bloque Fat-Tree k=4

ID	Tipo de ensayo	Comunicación evaluada	Tráfico generado	Objetivo metodológico
T1	Conectividad local	*Hosts* conectados al mismo *switch* de borde, por ejemplo `h1`→`h2`.	`ping` e `iperf3` TCP punto a punto.	Establecer una referencia mínima de latencia y *throughput* sin atravesar capas superiores de la topología.
T2	Comunicación intra-pod	*Hosts* situados en el mismo *pod*, pero conectados a *switches* de borde distintos, por ejemplo `h1`→`h3`.	`ping` e `iperf3` TCP punto a punto.	Evaluar el tránsito borde–agregación–borde dentro de un mismo *pod* y comprobar el comportamiento al añadir un salto jerárquico.
T3	Comunicación inter-pod	*Hosts* situados en *pods* distintos, por ejemplo `h1`→`h16`.	`ping` e `iperf3` TCP punto a punto.	Evaluar el camino completo borde–agregación–*core*–agregación–borde y validar el escenario representativo de mayor profundidad topológica.
T4	Tráfico concurrente balanceado	Varios pares origen–destino distribuidos entre *pods*.	Múltiples flujos `iperf3` TCP simultáneos.	Medir *throughput* agregado, reparto entre flujos y utilización de enlaces bajo carga concurrente controlada.

Continúa en la página siguiente

Tabla 7.1 – Continuación

ID	Tipo de ensayo	Comunicación evaluada	Tráfico generado	Objetivo metodológico
T5	Tráfico *hotspot* controlado	Varios orígenes hacia un mismo destino o hacia un mismo *pod*.	Múltiples flujos `iperf3` TCP simultáneos hacia un punto común.	Analizar el comportamiento del banco de pruebas ante concentración de tráfico y comprobar posibles desequilibrios de utilización.
T6	Escenario extendido condicionado	Comparación entre perfiles de *forwarding* o variantes estructurales si quedan implementadas de forma estable.	Mismo patrón de tráfico que en T4 o T5.	Permitir una comparación adicional solo si la implementación queda validada con evidencias reproducibles, evitando introducir resultados no consolidados.

Nota: La matriz separa los ensayos mínimos necesarios para caracterizar la instancia Fat-Tree k=4 de las extensiones condicionadas. De este modo, el Capítulo 7 define la metodología experimental sin anticipar resultados ni asumir como ejecutadas campañas que todavía deben validarse en el Capítulo 8.

A partir de esta matriz se definen escenarios de tráfico concretos. Los primeros escenarios cubren comunicaciones locales, intra-*pod* e inter-*pod*, de forma que sea posible observar el efecto de atravesar un número creciente de capas de la topología. Posteriormente se consideran escenarios concurrentes y de concentración de tráfico, que permiten evaluar el comportamiento del banco de pruebas cuando existen varios flujos activos de forma simultánea.

La Tabla 7.2 recoge los escenarios previstos, los pares de *hosts* representativos, la ruta lógica esperada y el propósito experimental asociado. Esta tabla vincula cada escenario con una interpretación topológica concreta, evitando que las pruebas se reduzcan a ejecuciones aisladas de tráfico sin contexto estructural.

Tabla 7.2: Escenarios de tráfico Fat-Tree k=4

ID	Escenario	Pares de *hosts* representativos	Ruta lógica esperada	Propósito experimental
T1	Comunicación local	`h1`→`h2`, ambos conectados al mismo *switch* de borde `e1`.	`h1`→`e1` `e1`→`h2`	Medir el comportamiento mínimo del plano de datos sin atravesar capas de agregación ni *core*. Sirve como referencia de latencia y *throughput* local.

T2	Comunicación intra-pod	`h1`→`h3`, situados en el mismo *pod*, pero conectados a *switches* de borde distintos.	`h1`→`e1`
			`e1`→`a1`
			`a1`→`e2`
			`e2`→`h3`
				Comprobar el tránsito dentro de un *pod* y evaluar el efecto de introducir la capa de agregación en la comunicación entre *hosts*.

T3	Comunicación inter-pod	`h1`→`h16`, situados en *pods* distintos.	`h1`→`e1`
			`e1`→`a1`
			`a1`→`c1`
			`c1`→`a7`
			`a7`→`e8`
			`e8`→`h16`
				Validar el camino jerárquico completo de la Fat-Tree, atravesando borde, agregación, *core*, agregación y borde.

T4	Tráfico concurrente distribuido	Varios pares origen–destino repartidos entre distintos *pods*, por ejemplo `h1`→`h16`, `h3`→`h14`, `h5`→`h12` y `h7`→`h10`.	Múltiples caminos inter-pod activos simultáneamente.	Evaluar *throughput* agregado, reparto entre flujos y utilización de enlaces cuando la red transporta varias comunicaciones concurrentes.
T5	*Hotspot* controlado	Varios *hosts* origen transmiten hacia un mismo *host* o hacia un mismo *pod* de destino, por ejemplo `h1`, `h3`, `h5` y `h7` hacia `h16`.	Convergencia de varios flujos hacia una zona común de la topología.	Analizar el comportamiento ante concentración de tráfico y detectar posibles cuellos de botella o desequilibrios de utilización.
T6	Escenario de comparación condicionada	Repetición de T4 o T5 bajo perfiles de *forwarding* alternativos, si estos quedan implementados y validados.	La ruta depende del perfil de *forwarding* aplicado.	Permitir una comparación adicional entre políticas de encaminamiento solo si existe evidencia técnica suficiente para defenderla.

Nota: Los escenarios T1–T3 cubren rutas representativas de complejidad creciente dentro de la topología Fat-Tree. Los escenarios T4 y T5 introducen carga concurrente y concentración de tráfico, respectivamente. El escenario T6 se mantiene condicionado para evitar que la metodología prometa comparativas que no hayan sido previamente implementadas y validadas de forma reproducible.

Los escenarios locales permiten comprobar el comportamiento mínimo de la topología cuando los *hosts* se encuentran conectados al mismo *switch* de borde. Los escenarios intra-*pod* añaden la capa de agregación, mientras que los escenarios inter-*pod* obligan a atravesar la jerarquía completa de borde, agregación, *core*, agregación y borde. Los escenarios concurrentes y *hotspot* introducen condiciones de carga más representativas para estudiar reparto entre flujos, concentración de tráfico y posibles desequilibrios.

El escenario de comparación condicionada debe tratarse con especial cautela. Su presencia en la metodología no implica que todos los perfiles alternativos estén garantizados como resultados finales. Solo se incorporará al Capítulo 8 si la implementación correspondiente queda validada con artefactos reproducibles.

7.3. Métricas experimentales previstas

La selección de métricas debe reflejar tanto el comportamiento de rendimiento como la trazabilidad de los ensayos. Sin embargo, no todas las métricas tienen el mismo grado de consolidación. Algunas proceden directamente de herramientas ya utilizadas, como

`iperf3` o `ping`; otras son derivadas de los valores por flujo; y otras dependen de que la captura de contadores o campañas específicas quede automatizada de forma fiable.

La Tabla 7.3 clasifica las métricas previstas según su uso metodológico. Esta clasificación evita asumir como disponibles medidas que todavía dependen de la ejecución experimental o de campañas adicionales.

Tabla 7.3: Métricas experimentales previstas en Fat-Tree k=4

Métrica	**Uso previsto**	**Fuente de medida**	**Interpretación experimental**
Throughput por flujo	Principal	Salida JSON de `iperf3`.	Representa la tasa efectiva alcanzada por cada comunicación origen–destino. Permite comparar rutas locales, intra-pod e inter-pod y detectar posibles penalizaciones entre flujos.
Throughput agregado	Principal	Agregación de varios flujos `iperf3` simultáneos.	Mide la capacidad total transportada por la topología durante escenarios concurrentes. Será una de las métricas centrales en los ensayos de carga distribuida.
Throughput medio por flujo	Derivada	Cálculo a partir de los valores de *throughput* por flujo y, cuando proceda, del cociente entre *throughput* agregado y número de flujos activos.	Permite normalizar el rendimiento agregado y comprobar si la capacidad obtenida se reparte de forma razonable entre las comunicaciones activas.
RTT medio	Principal	Salida de `ping`.	Resume la latencia extremo a extremo observada entre *hosts*. Se utilizará como indicador básico de retardo en escenarios locales, intra-pod e inter-pod.
RTT máximo	Complementaria	Salida de `ping`.	Permite identificar incrementos puntuales de latencia durante una prueba. Se interpretará con cautela, ya que puede estar afectado por la variabilidad propia del entorno emulado.
Utilización de enlace	Condicionada	Contadores de interfaces OVS o datos procesados a partir de capturas de ejecución.	Sirve para analizar reparto de carga, concentración de tráfico y posibles cuellos de botella. Solo se incorporará como métrica de resultados si la captura de contadores queda automatizada y validada de forma reproducible.
Índice de Jain	Derivada condicionada	Cálculo a partir del *throughput* por flujo en escenarios con múltiples flujos concurrentes.	Cuantifica la equidad del reparto entre flujos. Valores cercanos a 1 indican reparto más equilibrado; valores inferiores reflejan mayor desigualdad. Solo tiene sentido en escenarios concurrentes.
Pérdida de paquetes	Condicionada	Salida de `ping` o de pruebas UDP si se emplean.	Permite comprobar si existen pérdidas durante la validación. En pruebas TCP no será una métrica principal, ya que el propio protocolo reacciona ante pérdidas y congestión.
Jitter	Opcional	Pruebas UDP con `iperf3`, si se ejecutan.	Mide la variación temporal del retardo. Solo se incluirá en el Capítulo 8 si se realiza una campaña UDP específica y los resultados obtenidos son consistentes.

Nota: La tabla distingue entre métricas principales, derivadas, condicionadas y opcionales. Esta clasificación evita asumir como disponibles medidas que todavía dependen de la automatización experimental o de campañas específicas. En todos los casos, la interpretación tendrá en cuenta que el banco de pruebas se ejecuta sobre un entorno emulado con Mininet/OVS, por lo que las magnitudes absolutas no deben extrapolarse directamente a una red física de centro de datos.

El *throughput* por flujo y el *throughput* agregado constituyen métricas principales para caracterizar la capacidad efectiva observada en los ensayos. En escenarios concurrentes, el *throughput* medio por flujo permite normalizar el rendimiento agregado y comprobar si la capacidad obtenida se reparte de forma razonable entre las comunicaciones activas.

El RTT medio se utiliza como indicador básico de retardo extremo a extremo, especialmente en los escenarios local, intra-*pod* e inter-*pod*. El RTT máximo se considera una métrica complementaria, útil para detectar incrementos puntuales de latencia, pero debe interpretarse con cautela debido a la variabilidad propia del entorno emulado.

La utilización de enlaces, el índice de Jain, la pérdida de paquetes y el *jitter* requieren una interpretación más prudente. La utilización de enlaces solo debe incorporarse como resultado principal si los contadores se capturan de forma automatizada y reproducible. El índice de Jain solo tiene sentido cuando existen varios flujos concurrentes comparables. Las pérdidas y el *jitter* se mantendrán como métricas condicionadas u opcionales, especialmente si dependen de campañas UDP específicas.

Esta clasificación permite que el Capítulo 8 incorpore únicamente resultados respaldados por artefactos trazables. Si una métrica no puede reconstruirse a partir de datos brutos o procesados, no debe aparecer como conclusión cuantitativa principal.

7.4. Procedimiento experimental y pipeline metodológico

Además de definir escenarios y métricas, es necesario establecer una secuencia reproducible de ejecución. El objetivo es evitar que los resultados dependan de pasos manuales no documentados, de ejecuciones mezcladas o de artefactos incompletos. Para ello, el procedimiento experimental debe recorrer de forma ordenada la preparación del entorno, el despliegue de la topología, la conexión con la controladora, la instalación del *forwarding*, la validación previa, la ejecución de tráfico, la captura de datos y el procesado posterior.

La Figura 7.1 resume visualmente este flujo metodológico. El *pipeline* se organiza en cuatro bloques: diseño experimental, preparación del entorno, ejecución y captura, y procesado y evidencias. Esta representación conecta las decisiones metodológicas del capítulo con la generación posterior de resultados reproducibles.

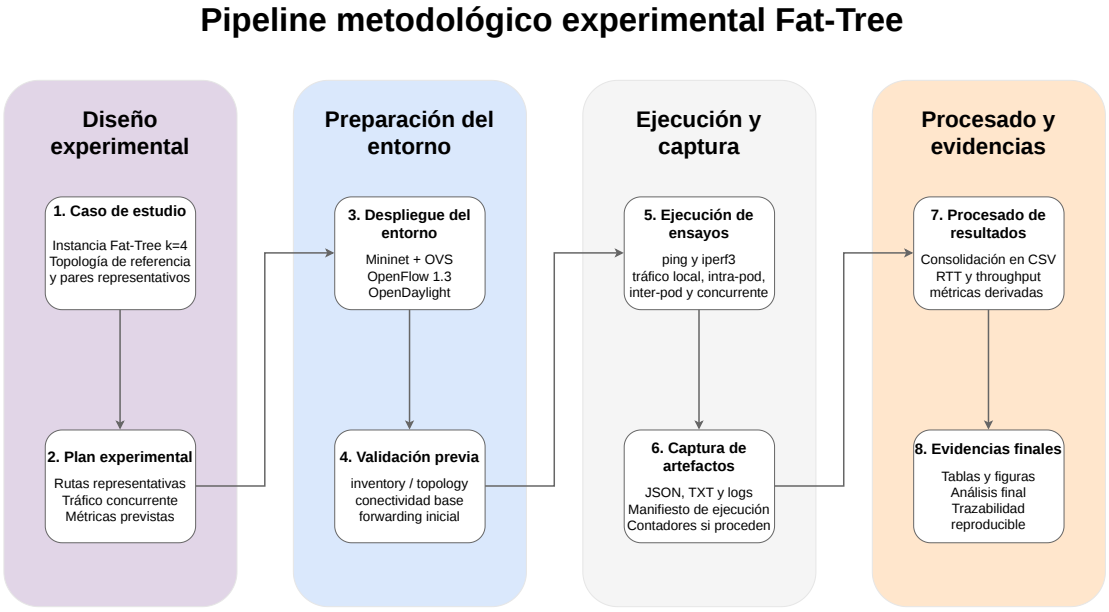

Figura 7.1: Pipeline metodológico experimental para la evaluación de Fat-Tree k=4.

El *pipeline* parte de la definición del caso de estudio y del plan experimental. A continuación, se prepara el entorno de ejecución mediante Mininet, Open vSwitch y OpenDaylight. Una vez desplegada la topología, se realiza una validación previa del inventario, la topología descubierta, la conectividad base y el *forwarding* inicial. Solo después de esta prevalidación se ejecutan los ensayos de tráfico y se capturan los artefactos necesarios para el análisis.

La Tabla 7.4 desarrolla este flujo como un procedimiento experimental paso a paso. Cada fase se acompaña de un criterio de aceptación, lo que permite distinguir entre una ejecución válida y una ejecución que no debería emplearse para generar resultados finales.

Tabla 7.4: Procedimiento experimental de Fat-Tree k=4

Paso	Fase	Acción prevista	Criterio de aceptación
1	Preparación del entorno	Arrancar OpenDaylight, comprobar disponibilidad de RESTCONF y verificar que el entorno de trabajo se ejecuta desde la raíz del proyecto.	La controladora debe estar accesible y el entorno debe encontrarse en un estado operativo conocido antes de iniciar Mininet.
2	Despliegue de la topología	Instanciar la topología Fat-Tree k=4 mediante el generador parametrizable desarrollado sobre Mininet.	Deben aparecer los *switches*, *hosts* y enlaces esperados para la instancia k=4, manteniendo la nomenclatura y el direccionamiento definidos en el Capítulo 6.
3	Conexión SDN	Comprobar que los *bridges* Open vSwitch quedan configurados con OpenFlow 1.3 y conectados a la controladora OpenDaylight.	OpenDaylight debe descubrir los nodos OpenFlow esperados y el inventario operacional debe ser coherente con la topología desplegada.

Continúa en la página siguiente

Tabla 7.4 – Continuación

Paso	Fase	Acción prevista	Criterio de aceptación
4	Aplicación de *forwarding*	Instalar el *forwarding* inicial necesario para ejecutar los escenarios de ensayo definidos en la metodología.	Las reglas deben quedar presentes en el *datapath* de Open vSwitch y permitir comunicación en los caminos representativos definidos.
5	Validación previa	Ejecutar pruebas de conectividad y comprobaciones mínimas de *datapath* antes de generar tráfico de rendimiento.	La validación previa debe confirmar conectividad entre *hosts* representativos y ausencia de fallos básicos de instalación o descubrimiento.
6	Ejecución de ensayos	Lanzar las pruebas de `ping` e `iperf3` según el escenario correspondiente: local, intra-pod, inter-pod o concurrente.	Cada ensayo debe generar salidas asociadas al escenario ejecutado, sin mezclar resultados de ejecuciones distintas.
7	Captura de artefactos	Guardar salidas de consola, JSON de `iperf3`, resultados de `ping`, manifiestos de ejecución y, si procede, contadores de interfaces o *dumps* de *flows*.	Los artefactos deben quedar almacenados en una estructura de directorios identificable y trazable respecto al escenario ejecutado.
8	Procesado de resultados	Transformar las salidas brutas en CSV o tablas intermedias, calculando únicamente las métricas realmente disponibles en los artefactos generados.	El procesado debe conservar la correspondencia entre escenario, repetición, pares de *hosts* y métricas calculadas.
9	Revisión de coherencia	Comprobar que los valores procesados son consistentes con el escenario ejecutado y que no existen resultados incompletos, mezclados o no reproducibles.	Solo se incorporarán al Capítulo 8 los resultados que tengan artefactos asociados y puedan justificarse metodológicamente.
10	Generación de evidencias finales	Crear las tablas y figuras finales que se integrarán en la memoria, manteniendo una separación clara entre datos brutos, datos procesados y material gráfico.	Las evidencias finales deben permitir reconstruir el origen de cada resultado y sostener una interpretación prudente en el Capítulo 8.

Nota: El procedimiento se plantea como una secuencia reproducible desde la preparación del entorno hasta la generación de evidencias finales. Su objetivo es evitar que los resultados experimentales dependan de ejecuciones manuales no trazables o de métricas que no hayan sido capturadas de forma verificable.

Este procedimiento refuerza la reproducibilidad del banco de pruebas. Antes de medir rendimiento, debe confirmarse que la controladora está accesible, que la topología se instancia correctamente, que los *bridges* Open vSwitch se conectan mediante OpenFlow 1.3 y que el *forwarding* instalado permite comunicación en los caminos representativos. Solo entonces tiene sentido lanzar pruebas de `ping` o `iperf3` asociadas a los escenarios definidos.

La captura de artefactos forma parte del procedimiento, no una tarea posterior opcional. Las salidas JSON de `iperf3`, los resultados de `ping`, los manifiestos de ejecución, los logs y, si

procede, los *dumps* o contadores de Open vSwitch deben conservarse de forma organizada. Esta decisión permite reconstruir el origen de cada valor presentado posteriormente.

7.5. Tratamiento de resultados, validez y limitaciones

El tratamiento de resultados debe mantener una separación clara entre datos brutos, datos procesados, tablas finales y figuras. Esta separación es necesaria para garantizar que cada resultado incluido en la memoria pueda rastrearse hasta una ejecución concreta y hasta los artefactos generados durante ella.

La Tabla 7.5 resume el tratamiento previsto para cada tipo de entrada. El criterio general es conservador: solo se calcularán métricas a partir de artefactos realmente disponibles y trazables.

Tabla 7.5: Tratamiento de resultados Fat-Tree k=4

Entrada	**Procesado previsto**	**Salida generada**	**Uso en la memoria**
Manifiesto de ejecución	Registrar escenario, repetición, pares origen–destino, fecha de ejecución y parámetros principales de la prueba.	Fichero de manifiesto asociado a cada ejecución o campaña experimental.	Permite reconstruir qué ensayo se ejecutó, bajo qué condiciones y con qué configuración.
Salida JSON de `iperf3`	Extraer *throughput* por flujo y, en escenarios concurrentes, agregar los valores por escenario y repetición.	CSV intermedio con *throughput* por flujo y resumen agregado por escenario.	Base cuantitativa para las tablas y figuras de rendimiento del Capítulo 8.
Salida de `ping`	Extraer pérdida de paquetes y resumen RTT cuando la salida incluya estadísticas válidas.	CSV o tabla intermedia con RTT mínimo, medio, máximo y pérdida observada.	Permite documentar la latencia básica y comprobar que los ensayos no se apoyan solo en *throughput*.
Datos por flujo en escenarios concurrentes	Calcular métricas derivadas únicamente cuando existan varios flujos medidos de forma comparable.	*Throughput* medio por flujo e índice de Jain si los datos disponibles lo permiten.	Sirve para analizar reparto entre comunicaciones activas sin asumir equidad cuando no hay datos suficientes.

Continúa en la página siguiente

Tabla 7.5 – Continuación

Entrada	**Procesado previsto**	**Salida generada**	**Uso en la memoria**
Dumps de *flows* o contadores OVS	Conservarlos como artefactos de inspección del *datapath*. Solo se convertirán en métricas si la captura queda automatizada y validada.	Ficheros TXT de soporte y, en su caso, CSV derivado de utilización.	Refuerzan la trazabilidad técnica. No se usarán como métrica principal si no existe captura reproducible.
CSV procesados	Revisar coherencia de unidades, escenarios, repeticiones, pares de *hosts* y valores ausentes.	CSV finales limpios y tablas resumen.	Evitan mezclar resultados incompletos o no comparables dentro de la interpretación final.
Figuras generadas	Crear gráficas solo a partir de CSV finales y no directamente desde salidas manuales aisladas.	Figuras en PDF/PNG/SVG según proceda.	Aseguran que las figuras del Capítulo 8 proceden de datos procesados de forma trazable.
Tablas finales	Integrar únicamente resultados con artefactos asociados y correspondencia clara con la campaña ejecutada.	Tablas LaTeX de resultados.	Permiten presentar resultados de forma compacta, verificable y coherente con la metodología.

Nota: El tratamiento de resultados se define de forma conservadora: solo se calcularán métricas a partir de artefactos realmente generados y trazables. Las medidas condicionadas, como la utilización de enlaces a partir de contadores OVS, solo se incorporarán a los resultados si la captura queda automatizada y puede reproducirse de forma consistente.

Las salidas JSON de `iperf3` se utilizarán para extraer *throughput* por flujo y, cuando existan escenarios concurrentes, valores agregados por escenario y repetición. Las salidas de `ping` permitirán obtener estadísticas de RTT y pérdida de paquetes cuando la salida incluya información válida. Los datos por flujo podrán emplearse para calcular métricas derivadas, como *throughput* medio por flujo o índice de Jain, únicamente si existen varios flujos medidos de forma comparable.

Los *dumps* de *flows* y contadores de Open vSwitch deben conservarse como evidencias de inspección del *datapath*. Sin embargo, solo se transformarán en métricas de utilización si la captura queda automatizada y validada de forma reproducible. Esta distinción evita presentar como resultado principal una medida que no pueda reconstruirse desde artefactos procesables.

Además del tratamiento de resultados, es necesario explicitar las limitaciones metodológicas de la evaluación. La Tabla 7.6 recoge los principales riesgos de validez y los criterios aplicados para limitar su impacto en la interpretación.

Tabla 7.6: Validez experimental y limitaciones de Fat-Tree k=4

Aspecto	**Riesgo metodológico**	**Criterio aplicado**	**Implicación para la interpretación**
Entorno emulado	Mininet y Open vSwitch se ejecutan sobre una misma máquina física, por lo que las medidas pueden depender de CPU, memoria, planificación del sistema operativo y carga local.	Interpretar los resultados como medidas obtenidas en un banco de pruebas emulado y no como prestaciones absolutas de una red física de centro de datos.	Las conclusiones deben centrarse en comparaciones internas y tendencias observadas dentro del mismo entorno experimental.
Escala de la topología	La instancia experimental se fija en Fat-Tree k=4, que es manejable en el entorno local pero no representa por sí sola una infraestructura de gran escala.	Usar k=4 como caso de estudio reproducible y complementar la discusión con la escalabilidad estructural ya definida en el Capítulo 6.	No se deben extrapolar directamente los valores absolutos a topologías Fat-Tree de mayor tamaño.
Forwarding aplicado	Las rutas dependen de las reglas OpenFlow instaladas y no de un cálculo automático completo de rutas *multipath* por parte de la controladora.	Describir explícitamente el perfil de *forwarding* utilizado en cada ensayo y evitar atribuir a OpenDaylight decisiones de encaminamiento que no se hayan implementado.	Los resultados evalúan el banco de pruebas y la política de *forwarding* aplicada, no un sistema autónomo de optimización SDN.
Variabilidad de TCP	Las pruebas con `iperf3` TCP pueden verse afectadas por mecanismos internos del protocolo, planificación de procesos y condiciones instantáneas del entorno emulado.	Registrar artefactos por ejecución y comparar únicamente ensayos obtenidos bajo condiciones equivalentes de topología, tráfico y configuración.	Pequeñas diferencias de *throughput* deben interpretarse con cautela, especialmente cuando el número de repeticiones sea reducido.
Latencia medida con `ping`	El RTT en un entorno local emulado puede alcanzar valores muy bajos y sensibles a variabilidad puntual del sistema.	Usar el RTT como indicador básico de conectividad y retardo relativo, no como medida representativa de latencia física de red.	La latencia complementa la interpretación del *throughput*, pero no debe convertirse en el único argumento de rendimiento.
Métricas condicionadas	Algunas métricas, como utilización de enlaces o reparto exacto de carga, dependen de que la captura de contadores quede automatizada y validada.	Incorporar esas métricas solo si existen artefactos reproducibles y procesado trazable; en caso contrario, mantenerlas como evidencias de soporte.	No se deben presentar como resultados principales métricas que no puedan reconstruirse a partir de los artefactos guardados.
Escenarios de tráfico	Los escenarios local, intra-pod, inter-pod, concurrente y *hotspot* representan patrones controlados, pero no cubren todos los comportamientos posibles de un centro de datos real.	Definir los escenarios como casos representativos de complejidad creciente y no como una caracterización exhaustiva de tráfico de producción.	Las conclusiones deben limitarse a los patrones de tráfico ensayados y a las condiciones descritas en la metodología.

Continúa en la página siguiente

Tabla 7.6 – Continuación

Aspecto	Riesgo metodológico	Criterio aplicado	Implicación para la interpretación
Trazabilidad de resultados	Existe riesgo de mezclar salidas manuales, ejecuciones incompletas o resultados procedentes de configuraciones distintas.	Separar datos brutos, datos procesados, manifiestos, figuras y tablas, manteniendo correspondencia entre escenario, ejecución y artefactos.	Solo deben integrarse en el Capítulo 8 resultados que puedan rastrearse hasta una ejecución concreta y verificable.

Nota: La validez experimental se plantea desde una perspectiva conservadora. El objetivo no es demostrar prestaciones absolutas de una red física, sino evaluar de forma reproducible el comportamiento de una instancia Fat-Tree k=4 dentro del banco de pruebas SDN desarrollado. Por ello, las conclusiones deberán formularse en términos relativos, trazables y acotados al entorno experimental utilizado.

La principal limitación deriva del uso de un entorno emulado. Mininet y Open vSwitch se ejecutan sobre una misma máquina física, por lo que las medidas pueden depender de CPU, memoria, planificación del sistema operativo y carga local. Por este motivo, las conclusiones deberán centrarse en comparaciones internas, tendencias observadas y diferencias relativas bajo condiciones equivalentes, no en prestaciones absolutas extrapolables a hardware real.

Asimismo, las campañas Leaf-Spine y Fat-Tree no se interpretan como una comparación directa de capacidad absoluta entre topologías. El bloque Leaf-Spine se diseñó con enlaces ascendentes limitados a 100 Mbps para observar saturación controlada y uso de *uplinks* en una arquitectura reducida, mientras que la campaña Fat-Tree se orienta a validar una arquitectura jerárquica mediante *forwarding* instalado desde OpenDaylight RESTCONF. Por ello, la comparación entre ambos bloques es metodológica y estructural, no una comparación directa de *throughput* absoluto.

También debe considerarse la escala de la topología. La instancia k=4 es adecuada como caso de estudio manejable y reproducible dentro del entorno local, pero no representa por sí sola una infraestructura de centro de datos a gran escala. La escalabilidad estructural se ha documentado en el Capítulo 6, mientras que la evaluación experimental se acota a la instancia implementada.

Otra limitación importante está relacionada con el *forwarding* aplicado. Los resultados evaluarán el banco de pruebas y la política de encaminamiento (*forwarding*) instalada, no un sistema autónomo de optimización SDN. Por tanto, no debe atribuirse a OpenDaylight un cálculo automático de rutas *multipath* que no haya sido implementado explícitamente. En particular, la presencia de caminos alternativos en Fat-Tree no implica por sí sola ECMP completo ni balanceo automático: cualquier extensión *multipath* deberá considerarse válida únicamente si queda respaldada por instalación verificable de reglas o grupos, conectividad funcional y evidencias de contadores en el *datapath*.

La metodología definida en este capítulo establece un marco experimental reproducible para evaluar la topología Fat-Tree k=4 implementada en el banco de pruebas SDN. La matriz experimental, los escenarios de tráfico, las métricas previstas, el procedimiento de ejecución, el tratamiento de resultados y las limitaciones metodológicas delimitan el alcance de la evaluación y evitan extrapolaciones no justificadas. A partir de esta base, el Capítulo 8 podrá presentar los resultados obtenidos únicamente a partir de artefactos trazables y procesados de forma consistente.

8. RESULTADOS EXPERIMENTALES

Este capítulo presenta los resultados obtenidos durante la evaluación experimental de la topología Fat-Tree k=4 en el banco de pruebas SDN desarrollado en los capítulos anteriores. Tras la validación incremental del entorno y la consolidación de la topología Leaf-Spine como etapa metodológica intermedia, el objetivo de este capítulo es comprobar el funcionamiento de una arquitectura Fat-Tree de mayor riqueza jerárquica, manteniendo una metodología reproducible y trazable.

La campaña principal de resultados se basa el objetivo de este capítulo es comprobar el funcionamiento de una arquitectura Fat-Tree de mayor riqueza en la instalación de reglas de *forwarding* determinista mediante OpenDaylight RESTCONF. Esta decisión metodológica responde a una necesidad práctica y académica: disponer de una base experimental estable, verificable y defendible antes de abordar mecanismos más complejos como ECMP o reconfiguración dinámica. Por tanto, los resultados de este capítulo no deben interpretarse como una evaluación de balanceo automático, sino como una validación experimental de caminos deterministas instalados desde la controladora SDN.

La campaña utilizada como referencia principal se identifica como:

```
ch08_fattree_restconf_20260519_0957
```

En ella se evaluaron tres tipos de comportamiento: ensayos punto a punto, tráfico concurrente distribuido y tráfico *hotspot* hacia un único destino. Todos los escenarios fueron ejecutados sobre la misma topología Fat-Tree k=4, formada por 16 *hosts*, 8 *switches* de borde, 8 *switches* de agregación y 4 *switches core*. La controladora OpenDaylight se utilizó para instalar las reglas de encaminamiento mediante RESTCONF, mientras que Mininet y Open vSwitch proporcionaron el entorno de emulación y los contadores del plano de datos.

Debe recordarse que esta campaña Fat-Tree responde a un objetivo experimental distinto del bloque Leaf-Spine del Capítulo 5. En Leaf-Spine se utilizaron enlaces ascendentes limitados a 100 Mbps para observar saturación controlada y uso de *uplinks*; en este capítulo, en cambio, los valores de *throughput* se interpretan dentro de la campaña Fat-Tree RESTCONF y de las condiciones propias del entorno emulado. Por tanto, los resultados no deben leerse como una comparación directa de capacidad absoluta entre ambas topologías.

8.1. Alcance metodológico de la campaña RESTCONF

La campaña RESTCONF tiene como finalidad validar que el banco de pruebas es capaz de instalar, verificar y utilizar reglas OpenFlow generadas desde la controladora para transportar tráfico real en una topología Fat-Tree. La instalación se realizó mediante un script específico que genera los flujos necesarios para los pares experimentales planificados, los envía a OpenDaylight y posteriormente verifica su presencia en Open vSwitch mediante volcados del *datapath*.

Es importante delimitar con precisión el alcance de esta campaña. Las reglas instaladas implementan caminos deterministas para un conjunto concreto de pares y escenarios de tráfico. No se pretende validar conectividad universal entre todos los *hosts* ni ejecutar un `pingall` completo. Asimismo, la campaña no implementa ECMP, balanceo automático entre múltiples caminos ni reconfiguración dinámica basada en telemetría. Estos mecanismos quedan fuera del núcleo experimental principal y se consideran extensiones posteriores.

Esta delimitación también afecta a la interpretación de las métricas. Los valores obtenidos en la campaña RESTCONF se utilizan para comparar escenarios internos de la propia topología Fat-Tree, como comunicaciones locales, intra-*pod*, inter-*pod*, tráfico distribuido o *hotspot*. No se utilizan para establecer una jerarquía de rendimiento absoluto frente a la campaña Leaf-Spine, ya que ambos bloques fueron diseñados con condiciones y finalidades experimentales diferentes.

La Tabla 8.1 resume el estado de los bloques ejecutados durante la campaña: instalación de *forwarding*, preflight, ensayos punto a punto, tráfico concurrente distribuido y escenario *hotspot*.

Tabla 8.1: Estado de la campaña RESTCONF determinista

Bloque	Evidencia	Estado	OK/Total	Observación
Instalación de *forwarding*	Reglas RESTCONF instaladas y verificadas en Open vSwitch.	PASS	84/84	Todas las reglas planificadas quedaron presentes en el *datapath*.
Preflight de validación	Conectividad inicial, ARP estático y contadores de flujo.	PASS	6/6	Los pares representativos respondieron correctamente y los contadores reflejaron tráfico real.
Ensayos punto a punto	Pruebas local, intrapod e interpod.	PASS	9/9	Las tres clases de camino se ejecutaron con tres repeticiones válidas cada una.
Tráfico concurrente distribuido	Cuatro flujos TCP simultáneos sobre caminos distribuidos.	PASS	12/12	El precheck fue correcto y los flujos concurrentes completaron las repeticiones previstas.
Escenario *hotspot*	Cuatro flujos TCP concurrentes hacia `h16`.	PASS	12/12	Las reglas específicas del escenario fueron verificadas y los flujos finalizaron correctamente.

La instalación RESTCONF finalizó correctamente con 84 reglas instaladas y verificadas en OVS. Posteriormente, el preflight confirmó la conectividad de los pares experimentales, la existencia de entradas ARP estáticas y la presencia de contadores no nulos en los flujos relevantes. A partir de esta base se ejecutaron los ensayos punto a punto, concurrentes y *hotspot*.

8.2. Instalación y verificación del *forwarding* determinista

La topología Fat-Tree k=4 se arrancó desde Mininet utilizando el generador de topología desarrollado para el proyecto. Los *switches* se configuraron como Open vSwitch con soporte para OpenFlow 1.3 y se conectaron a OpenDaylight a través del puerto OpenFlow 6653. El Código 8.1 conserva el comando de arranque empleado.

```
cd ~/Escritorio/TFG/TFG-SDN

sudo -E mn \
  --custom tfg_sdn/mininet/topos/fattree.py \
  --topo fattree,k=4 \
  --controller=remote,ip=127.0.0.1,port=6653 \
  --switch ovsk,protocols=OpenFlow13 \
  --mac
```

Código 8.1: Arranque de la topología Fat-Tree k=4.

Tras el arranque, OpenDaylight descubrió los 20 nodos OpenFlow esperados, correspondientes a los 4 *switches core*, 8 de agregación y 8 de borde. La instalación de *forwarding* determinista se realizó mediante el script `ch08_install_restconf_forwarding.py`, que generó los flujos asociados a los caminos planificados y los instaló mediante RESTCONF. El Código 8.2 recoge la invocación utilizada.

```
python3 tfg_sdn/experiments/ch08_install_restconf_forwarding.py \
  --profile all \
  --cleanup-ovs
```

Código 8.2: Instalación del *forwarding* determinista mediante RESTCONF.

La verificación tuvo dos niveles. En primer lugar, se comprobó que las peticiones RESTCONF eran aceptadas por la controladora. En segundo lugar, se verificó que las reglas llegaban al *datapath* de OVS, evitando considerar como válida una regla que solo estuviera presente en el almacén de configuración de OpenDaylight. Esta doble comprobación es relevante porque en SDN no basta con generar una intención de control: es necesario comprobar su efecto en el plano de datos. El Código 8.3 resume el resultado de esta verificación.

```
Instalacion RESTCONF de forwarding determinista
status: PASS
n_flows: 84
n_installed_ok: 84
n_install_failed: 0

Resultado de verificacion en OVS
status: PASS
n_checks: 84
n_ok: 84
n_fail: 0
```

Código 8.3: Verificación del *forwarding* determinista en OVS.

El resultado fue satisfactorio: se instalaron y verificaron 84 reglas de *forwarding*, sin fallos de instalación ni de comprobación en OVS. Esta evidencia cierra el bloque de instalación RESTCONF y permite utilizar la campaña como base principal de resultados del capítulo.

8.3. Preflight de conectividad y contadores

Antes de ejecutar los ensayos de rendimiento se realizó un preflight específico. Este paso tuvo tres objetivos: limpiar posibles entradas ARP previas, instalar entradas ARP estáticas para los pares experimentales y comprobar conectividad básica mediante pings representativos.

La decisión de utilizar ARP estático es coherente con la metodología del capítulo. Las reglas RESTCONF instaladas incluyen descartes explícitos de ARP para evitar aprendizaje no controlado y mantener un comportamiento determinista. Por ello, las pruebas no deben depender de resolución ARP dinámica, sino de entradas previamente fijadas en los *hosts* implicados.

El preflight validó seis pares representativos:

- $h1 \rightarrow h2$, escenario local.
- $h1 \rightarrow h3$, escenario intra-*pod*.
- $h1 \rightarrow h16$, escenario inter-*pod*.
- $h3 \rightarrow h14$, escenario distribuido.
- $h5 \rightarrow h12$, escenario distribuido.
- $h7 \rightarrow h10$, escenario distribuido.

El Código 8.4 muestra la secuencia de preparación, ejecución dentro de Mininet y resumen posterior.

```
python3 tfg_sdn/experiments/ch08_restconf_preflight_validate.py \
  --campaign-id ch08_fattree_restconf_20260519_0957 \
  --action prepare

# Dentro de Mininet:
source runs/ch08_fattree/ch08_fattree_restconf_20260519_0957/preflight_restconf/mininet_commands.
    cli

python3 tfg_sdn/experiments/ch08_restconf_preflight_validate.py \
  --campaign-id ch08_fattree_restconf_20260519_0957 \
  --action summarize
```

Código 8.4: Ejecución del preflight RESTCONF.

Todos los pings finalizaron con 0 % de pérdida. Además, los contadores de los flujos relevantes mostraron paquetes y bytes no nulos después de las pruebas, lo que confirma que el tráfico atravesó realmente las reglas instaladas. El Código 8.5 recoge el resultado final del preflight.

```
Preflight RESTCONF
status: PASS
n_ping_ok: 6/6
arp_evidence_ok: True
flow_counter_evidence_ok: True
```

Código 8.5: Resultado del preflight RESTCONF.

8.4. Ensayos punto a punto

El primer bloque de rendimiento se centró en ensayos punto a punto. El objetivo fue comparar tres tipos de comunicación dentro de la topología Fat-Tree: comunicación local bajo el mismo *switch* de borde, comunicación intra-*pod* entre *switches* de borde del mismo *pod* y comunicación inter-*pod* atravesando la capa *core*.

Los escenarios fueron:

- Local: $h1 \rightarrow h2$.
- Intra-*pod*: $h1 \rightarrow h3$.
- Inter-*pod*: $h1 \rightarrow h16$.

Cada escenario se ejecutó con tres repeticiones válidas, obteniéndose un total de 9 resultados correctos. La Tabla 8.2 recoge el resumen de los ensayos, incluyendo el par evaluado, el número de repeticiones correctas, el *throughput*, el RTT medio y el camino determinista utilizado.

Tabla 8.2: Ensayos punto a punto RESTCONF

Escenario	Par	OK	Throughput medio	RTT medio	Camino determinista
Local	`h1` → `h2`	3/3	24,15 Gbps	0,093 ms	`h1-e1-h2`
Intrapod	`h1` → `h3`	3/3	21,64 Gbps	0,108 ms	`h1-e1-a1` `a1-e2-h3`
Interpod	`h1` → `h16`	3/3	19,60 Gbps	0,131 ms	`h1-e1-a1-c1` `c1-a7-e8-h16`

La Figura 8.1 muestra el RTT medio de los pings de control. Se observa un incremento progresivo del RTT desde el escenario local hasta el inter-*pod*. Aunque los valores absolutos son muy reducidos, como corresponde a un entorno emulado, la tendencia es coherente con la estructura jerárquica de Fat-Tree: el tráfico local requiere menos saltos que el tráfico intra-*pod*, y este a su vez menos que el tráfico inter-*pod*.

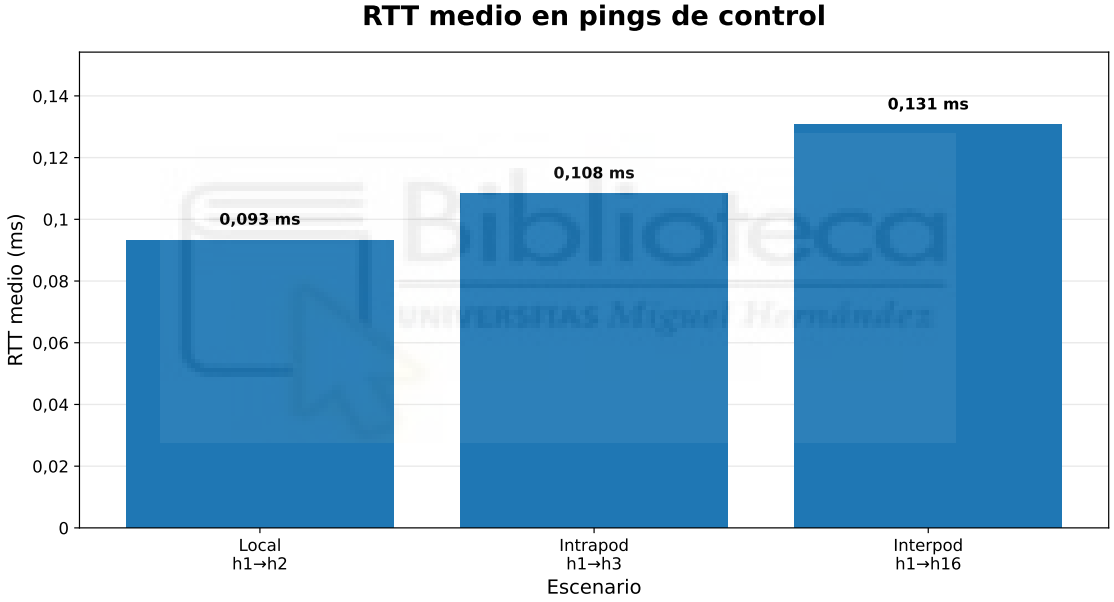

Figura 8.1: RTT medio en los ensayos punto a punto.

La Figura 8.2 presenta el *throughput* medio recibido en los tres escenarios. El escenario local alcanza aproximadamente 24,15 Gbps, el intra-*pod* 21,64 Gbps y el inter-*pod* 19,60 Gbps. Estos valores deben interpretarse como medidas internas de la campaña Fat-Tree RESTCONF en Mininet/OVS, no como capacidades físicas absolutas ni como magnitudes directamente comparables con el bloque Leaf-Spine limitado a 100 Mbps. Bajo esa lectura acotada, la reducción gradual resulta razonable, ya que los caminos más largos atraviesan más elementos virtuales y mayor complejidad en el plano de datos.

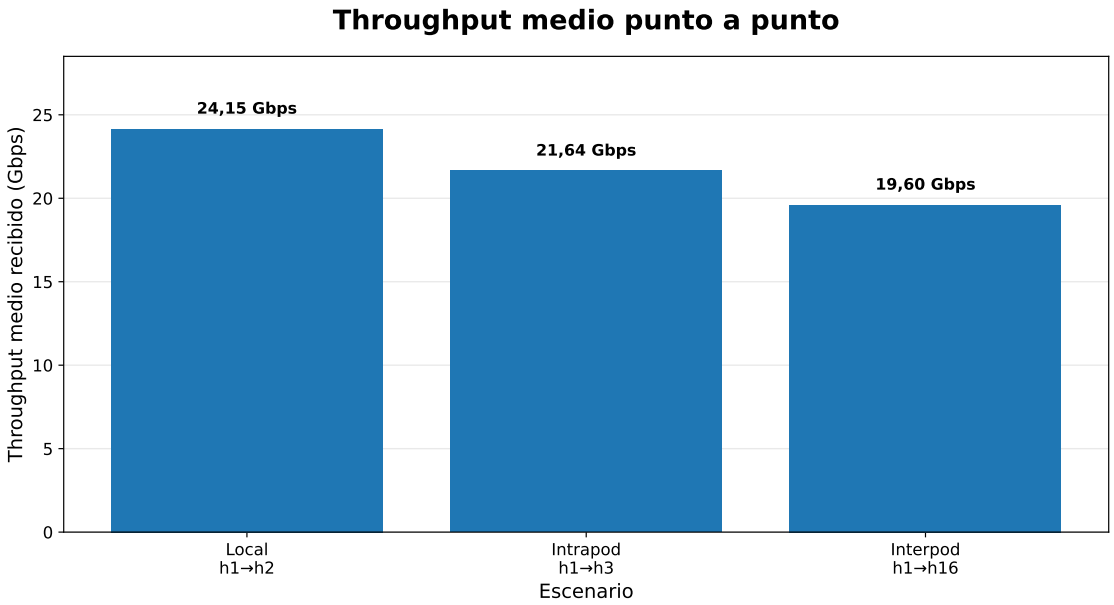

Figura 8.2: *Throughput* medio punto a punto.

Estos resultados no deben interpretarse como medidas absolutas de una red física real, sino como indicadores comparativos dentro del mismo entorno de emulación. Bajo esa lectura, el bloque punto a punto confirma que los caminos deterministas instalados por RESTCONF transportan tráfico de forma estable y que la jerarquía de la topología se refleja en las métricas obtenidas.

8.5. Escenario concurrente distribuido

El segundo bloque experimental introdujo tráfico concurrente distribuido. En este caso se ejecutaron cuatro flujos TCP simultáneos entre pares distribuidos por la topología:

- $h1 \rightarrow h16$.
- $h3 \rightarrow h14$.
- $h5 \rightarrow h12$.
- $h7 \rightarrow h10$.

El objetivo era comprobar si el *forwarding* determinista mantenía un comportamiento estable cuando varios caminos eran utilizados al mismo tiempo. Se realizaron tres repeticiones, con cuatro flujos por repetición, dando lugar a 12 resultados válidos. La Tabla 8.3 resume los valores por flujo y conserva el camino determinista asociado a cada par.

Tabla 8.3: Escenario concurrente distribuido

Flujo	Par	OK	Throughput medio	Rango	Camino determinista
Flujo desde h1	h1 → h16	3/3	7,32 Gbps	7,00–7,80 Gbps	h1-e1-a1-c1 c1-a7-e8-h16
Flujo desde h3	h3 → h14	3/3	8,00 Gbps	7,45–8,53 Gbps	h3-e2-a2-c4 c4-a8-e7-h14
Flujo desde h5	h5 → h12	3/3	7,33 Gbps	7,15–7,61 Gbps	h5-e3-a3-c2 c2-a5-e6-h12
Flujo desde h7	h7 → h10	3/3	7,69 Gbps	7,06–8,05 Gbps	h7-e4-a4-c3 c3-a6-e5-h10

La Figura 8.3 muestra el *throughput* agregado por repetición. Los valores se sitúan alrededor de 30 Gbps, con aproximadamente 29,75 Gbps en la primera repetición, 29,89 Gbps en la segunda y 31,40 Gbps en la tercera. Esta estabilidad entre repeticiones indica que la campaña concurrente no presenta fallos evidentes de conectividad ni degradaciones extremas.

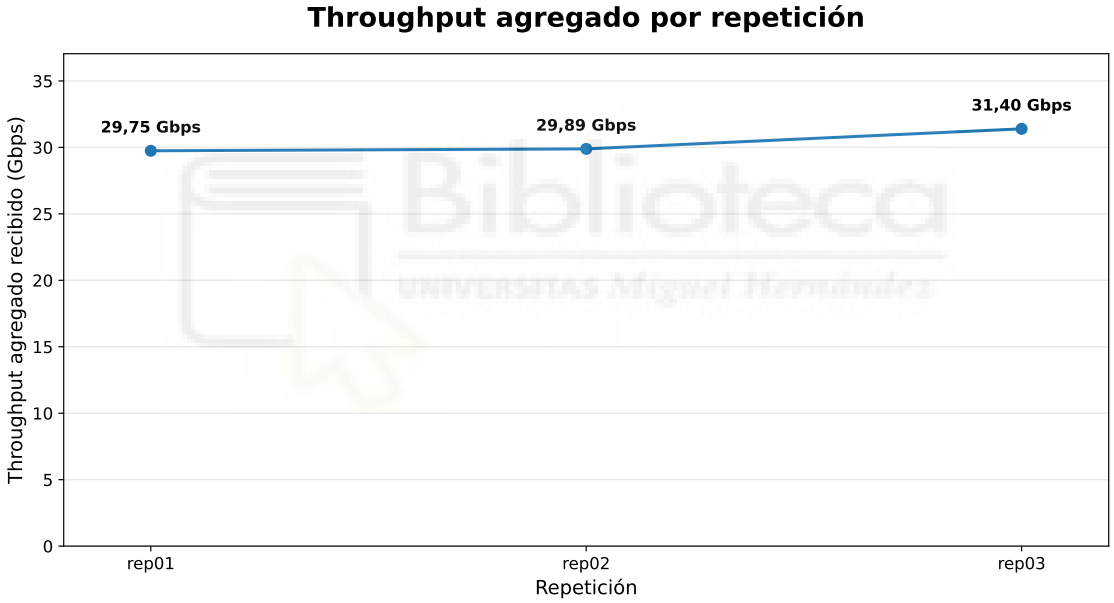

Figura 8.3: *Throughput* agregado en tráfico distribuido.

La Figura 8.4 descompone el rendimiento medio por flujo. Los cuatro flujos se sitúan aproximadamente entre 7,32 y 8,00 Gbps. No se observa un desequilibrio severo entre ellos, lo que sugiere que los caminos deterministas utilizados soportan correctamente el tráfico simultáneo planificado.

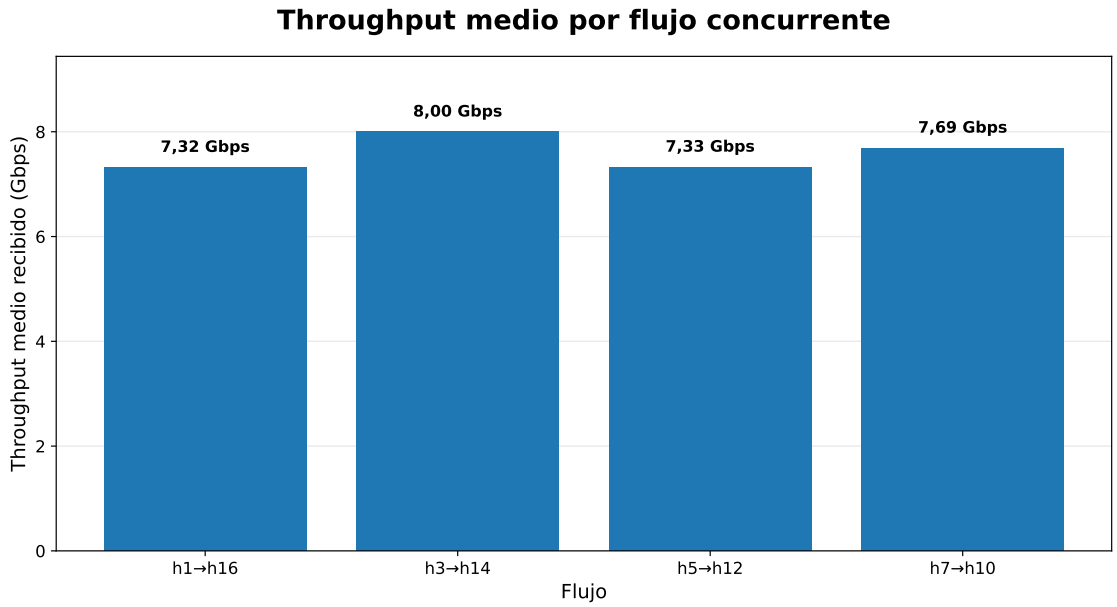

Figura 8.4: *Throughput* por flujo en tráfico distribuido.

Este bloque permite afirmar que la instalación RESTCONF no solo funciona en pruebas aisladas, sino también en un escenario con varios flujos concurrentes. Sin embargo, no debe confundirse esta observación con balanceo ECMP: los caminos fueron seleccionados de forma determinista por la planificación de flujos.

8.6. Escenario *hotspot*

El tercer bloque experimental evaluó un escenario *hotspot*, en el que varios emisores generan tráfico hacia un mismo destino. En este caso, el destino común fue $h16$, y los flujos ejecutados fueron:

- $h1 \rightarrow h16$.
- $h3 \rightarrow h16$.
- $h5 \rightarrow h16$.
- $h7 \rightarrow h16$.

Para este escenario fue necesario instalar reglas adicionales específicas, con prioridad superior a las reglas base, ya que el patrón *hotspot* no estaba cubierto completamente por los caminos T1–T4. Estas reglas se instalaron también mediante OpenDaylight RESTCONF y se verificaron en OVS antes de ejecutar el tráfico.

El objetivo del *hotspot* no es demostrar reconfiguración dinámica ni balanceo automático, sino observar el comportamiento de la topología cuando varios flujos convergen hacia un

mismo *host* destino bajo *forwarding* determinista. La Tabla 8.4 resume los resultados por flujo y el camino determinista utilizado por cada comunicación.

Tabla 8.4: Escenario *hotspot*

Flujo	Par	OK	Throughput medio	Rango	Camino determinista
Flujo desde `h1`	`h1` → `h16`	3/3	7,44 Gbps	6,59–8,24 Gbps	`e1-a1-c1` `c1-a7-e8`
Flujo desde `h3`	`h3` → `h16`	3/3	7,59 Gbps	7,22–7,84 Gbps	`e2-a2-c4` `c4-a8-e8`
Flujo desde `h5`	`h5` → `h16`	3/3	7,33 Gbps	6,99–7,86 Gbps	`e3-a3-c2` `c2-a7-e8`
Flujo desde `h7`	`h7` → `h16`	3/3	7,54 Gbps	7,32–7,73 Gbps	`e4-a4-c3` `c3-a8-e8`

La Figura 8.5 muestra el *throughput* agregado por repetición. Las dos primeras repeticiones se mantienen en torno a 30 Gbps, mientras que la tercera baja hasta aproximadamente 28,85 Gbps. Esta variabilidad es aceptable dentro del entorno de emulación y resulta coherente con un escenario de concentración de tráfico hacia un mismo destino.

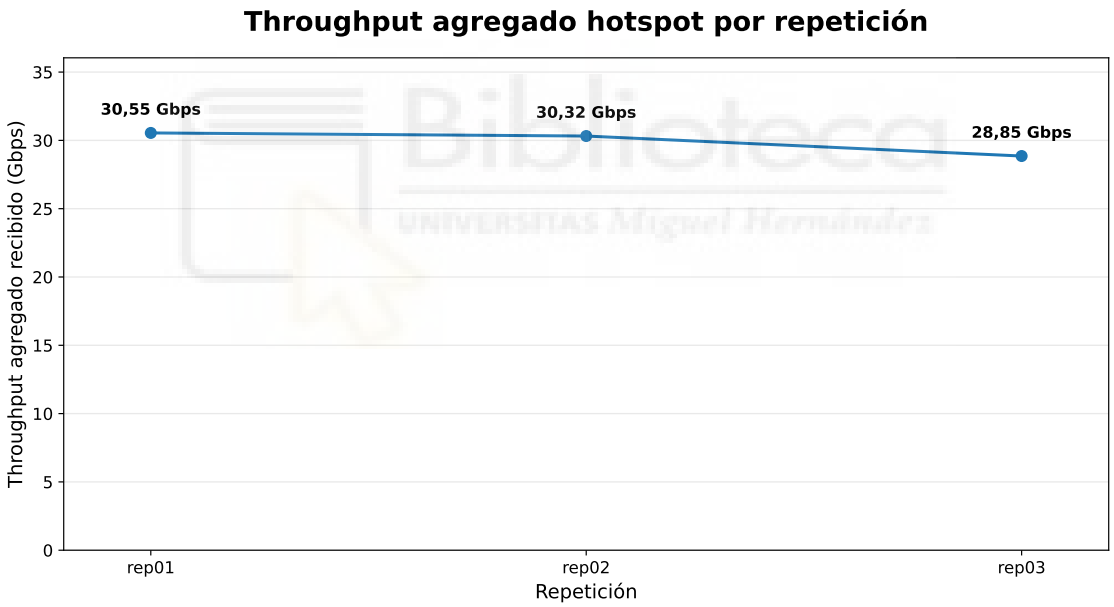

Figura 8.5: *Throughput* agregado en el escenario *hotspot*.

La Figura 8.6 presenta el *throughput* medio recibido por flujo. Los cuatro flujos se sitúan entre 7,33 y 7,59 Gbps aproximadamente. Esta proximidad entre valores indica que, pese a la concentración en $h16$, el escenario mantiene un reparto relativamente equilibrado entre los emisores dentro de la planificación determinista utilizada.

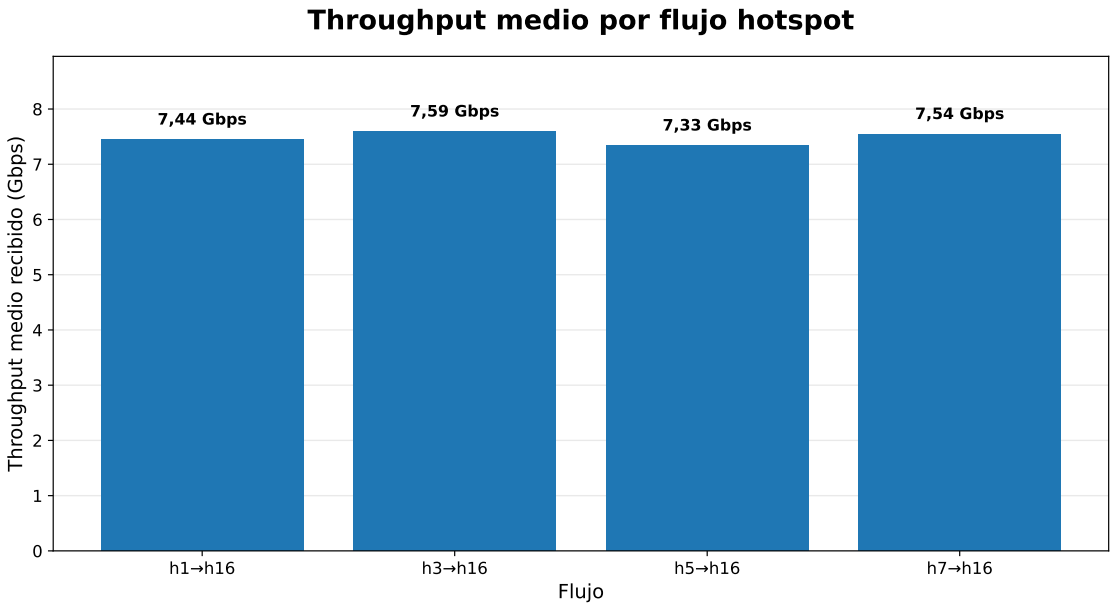

Figura 8.6: *Throughput* por flujo en el escenario *hotspot*.

El bloque *hotspot* aporta una evidencia relevante para el capítulo, ya que introduce un patrón de tráfico más exigente que los ensayos punto a punto y distinto del tráfico distribuido. Aunque el comportamiento sigue siendo estable, la ligera reducción del agregado medio frente al caso distribuido sugiere que la concentración de tráfico puede afectar al rendimiento global.

8.7. Comparativa entre tráfico distribuido y *hotspot*

La Figura 8.7 compara el *throughput* agregado medio entre el escenario de tráfico concurrente distribuido y el escenario *hotspot*. El tráfico distribuido alcanza aproximadamente 30,34 Gbps agregados, mientras que el *hotspot* se sitúa alrededor de 29,91 Gbps.

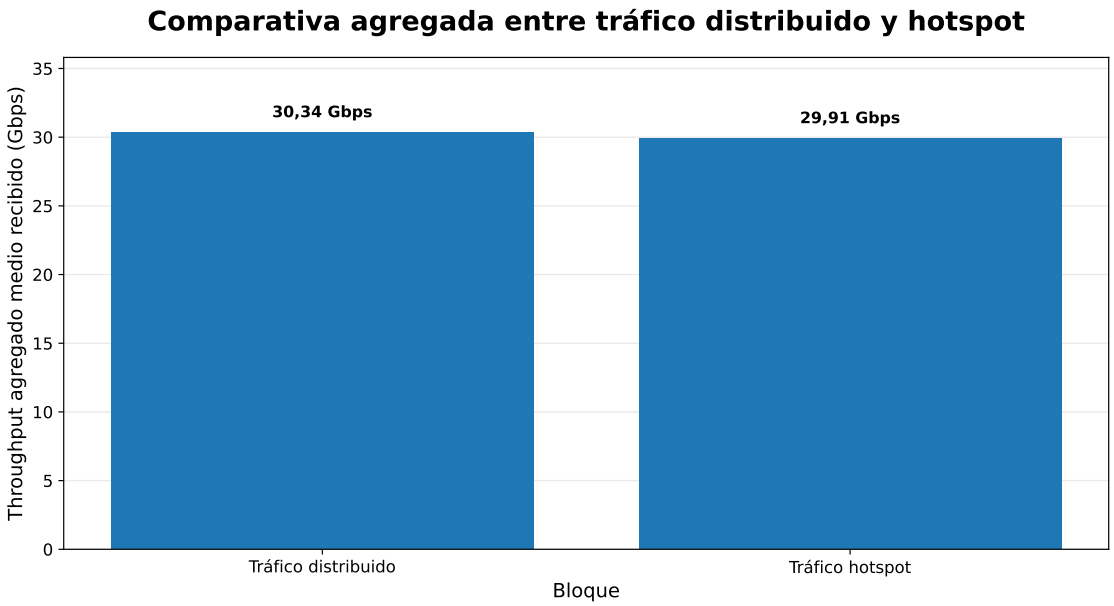

Figura 8.7: Comparativa de *throughput* agregado entre tráfico distribuido y *hotspot*.

La diferencia entre ambos escenarios es moderada, pero coherente. En el caso distribuido, los destinos y caminos están más repartidos, mientras que en el *hotspot* varios flujos convergen hacia el mismo *host* y su *switch* de borde asociado. Este patrón puede introducir una mayor presión sobre la zona final del camino, aunque en esta campaña no se observa una degradación severa.

La Tabla 8.5 resume de forma conjunta los resultados de *throughput* de la campaña, separando ensayos punto a punto, tráfico concurrente distribuido y escenario *hotspot*.

Tabla 8.5: Síntesis de *throughput* RESTCONF

Bloque	**Tipo**	**Elemento**	**Throughput medio**	**Agregado medio**	**Nota**
Ensayos punto a punto	Punto a punto	Comunicación local	24,15 Gbps	–	Flujo individual entre `h1` y `h2`.
Ensayos punto a punto	Punto a punto	Comunicación intrapod	21,64 Gbps	–	Flujo individual entre `h1` y `h3`.
Ensayos punto a punto	Punto a punto	Comunicación interpod	19,60 Gbps	–	Flujo individual entre `h1` y `h16`.
Tráfico concurrente	Distribuido	Agregado del escenario	7,59 Gbps	30,34 Gbps	Valor medio por flujo y agregado medio por repetición.
Tráfico *hotspot*	Concentrado hacia `h16`	Agregado del escenario	7,48 Gbps	29,91 Gbps	Valor medio por flujo y agregado medio por repetición.

Nota: En los ensayos punto a punto se muestra el valor de un flujo individual; en los escenarios concurrente y *hotspot*, el valor medio por flujo y el agregado medio por repetición.

Desde el punto de vista metodológico, esta comparación refuerza la validez del banco de pruebas: el sistema no solo ejecuta tráfico aislado, sino que permite construir escenarios diferenciados, capturar métricas, procesar resultados y obtener conclusiones interpreta-

bles. Al mismo tiempo, los resultados muestran que la topología Fat-Tree, incluso bajo *forwarding* determinista, soporta tráfico concurrente de manera estable en el entorno emulado.

8.8. Procesado de resultados

El procesado final de la campaña se realizó a partir de los artefactos generados durante la ejecución: salidas JSON de `iperf3`, resultados de `ping`, resúmenes de validación, volcados de *flows* y comprobaciones RESTCONF. El objetivo de este paso fue convertir las salidas brutas en CSV consolidados, tablas LaTeX y figuras finales para la memoria. El Código 8.6 muestra la invocación del script de procesado.

```
python3 tfg_sdn/experiments/ch08_process_restconf_results.py \
  --campaign-id ch08_fattree_restconf_20260519_0957
```

Código 8.6: Procesado de resultados de la campaña RESTCONF.

El proceso generó siete figuras en formatos PNG, SVG y PDF, así como seis archivos LaTeX de apoyo, de los cuales cinco corresponden a tablas integradas en el Capítulo 8. El resumen de procesado confirma que todos los bloques principales de la campaña fueron considerados válidos, tal como recoge el Código 8.7.

```
Procesado de resultados RESTCONF
================================================
Instalacion RESTCONF: PASS (84/84)
Preflight RESTCONF: PASS (6/6)
Ensayos punto a punto: PASS (9/9)
Escenario concurrente distribuido: PASS (12/12)
Escenario hotspot: PASS (12/12)

figures_generated: 7
figure_formats: PNG, SVG, PDF
latex_support_files_generated: 6
latex_tables_integrated: 5
status: PASS
```

Código 8.7: Resumen del procesado final RESTCONF.

Esta etapa es relevante porque mantiene la coherencia metodológica definida en el Capítulo 7: las figuras y tablas de resultados no se construyen directamente a partir de observaciones manuales, sino de artefactos procesados y trazables.

8.9. Limitaciones de la campaña

Aunque los resultados obtenidos son positivos, deben interpretarse dentro de los límites del banco de pruebas. En primer lugar, Mininet es un emulador y comparte los recursos de

la máquina anfitriona, por lo que los valores absolutos de *throughput* no deben extrapolarse directamente a una infraestructura física. La utilidad de las medidas reside principalmente en su comparación relativa dentro del mismo entorno y bajo la misma campaña experimental. En particular, las cifras de esta campaña Fat-Tree no deben compararse de forma absoluta con las obtenidas en Leaf-Spine, ya que allí se introdujo una limitación explícita de 100 Mbps en los enlaces ascendentes con finalidad didáctica y metodológica.

En segundo lugar, el *forwarding* utilizado es determinista. Esto permite trazabilidad y reproducibilidad, pero no evalúa la capacidad de la red para repartir tráfico automáticamente entre múltiples caminos equivalentes. Por tanto, no puede afirmarse que estos resultados demuestren ECMP en Fat-Tree.

En tercer lugar, la campaña no valida conectividad universal entre todos los pares de *hosts*. Las reglas se diseñaron para los escenarios experimentales planificados, y el uso de ARP estático forma parte de esa decisión metodológica. Esto evita comportamientos no controlados, pero limita el alcance de la conectividad evaluada.

Por último, el escenario *hotspot* no incorpora reconfiguración dinámica. Aunque permite observar el comportamiento ante concentración de tráfico, no incluye detección automática de congestión ni modificación de caminos en tiempo de ejecución.

Estas limitaciones no invalidan la campaña. Al contrario, delimitan correctamente lo que se ha demostrado: una instalación RESTCONF reproducible de *forwarding* determinista sobre Fat-Tree k=4, con tráfico real, métricas de rendimiento y evidencias de contadores en el plano de datos.

8.10. Conclusión del bloque RESTCONF

La campaña RESTCONF determinista constituye el núcleo experimental principal de este capítulo. Los resultados muestran que el banco de pruebas es capaz de arrancar una topología Fat-Tree k=4, conectar todos sus *switches* a OpenDaylight, instalar reglas OpenFlow mediante RESTCONF, verificar su presencia en OVS y ejecutar tráfico TCP sobre caminos deterministas.

Los ensayos punto a punto reflejan una tendencia coherente con la jerarquía de la topología: los caminos locales presentan menor RTT y mayor *throughput* que los caminos más largos. Los ensayos concurrentes distribuidos muestran un agregado estable alrededor de 30 Gbps, mientras que el escenario *hotspot* mantiene un rendimiento similar, aunque ligeramente inferior en promedio debido a la concentración de tráfico hacia $h16$.

En conjunto, la campaña demuestra una base sólida, trazable y reproducible para el Capítulo 8. Sobre esta base, resulta razonable plantear una fase adicional de demostración ECMP limitada, entendida como extensión exploratoria y no como sustitución de la campaña principal ya validada.

8.11. Extensión exploratoria: demostrador ECMP limitado

Tras cerrar la campaña principal basada en *forwarding* determinista instalado mediante OpenDaylight RESTCONF, se desarrolló una extensión exploratoria orientada a comprobar si era posible activar un mecanismo *multipath* controlado sobre la topología Fat-Tree k=4. Esta extensión no sustituye a la campaña determinista anterior, sino que la complementa con una prueba acotada basada en grupos OpenFlow de tipo *select*.

La motivación de esta extensión procede del bloque Leaf-Spine del Capítulo 5, donde se comprobó que el uso de caminos equivalentes permitía aprovechar mejor la redundancia disponible que un perfil de camino único. Sin embargo, trasladar ECMP completo a una Fat-Tree implica una complejidad mayor: múltiples *pods*, *switches* de agregación, *switches core*, rutas simétricas, selección de *buckets* y verificación de contadores en varios niveles de la jerarquía. Por este motivo, se planteó un demostrador limitado centrado exclusivamente en el tráfico entre $h1$ y $h16$.

El objetivo de esta extensión no es demostrar ECMP completo en toda la Fat-Tree, ni implementar balanceo dinámico, ni validar conectividad universal entre todos los *hosts*. El objetivo es más concreto: comprobar si, sobre una ruta inter-*pod* representativa, es posible activar dos ramas equivalentes y verificar mediante contadores que ambas ramas son utilizadas por tráfico real.

Metodológicamente, el demostrador se dividió en dos etapas. En primer lugar, se programó directamente el plano de datos mediante `ovs-ofctl`, con el fin de comprobar la viabilidad del mecanismo en Open vSwitch. En segundo lugar, se trasladó el mismo concepto a OpenDaylight RESTCONF, comprobando si la controladora podía instalar los grupos y los flujos necesarios para reproducir el demostrador sin depender únicamente de programación local directa sobre el *datapath*.

8.11.1. Validación inicial en el plano de datos

La primera etapa comparó dos configuraciones sobre el par $h1 \rightarrow h16$. La primera configuración forzó un único camino, utilizado como referencia *single_path*. La segunda configuración instaló grupos OpenFlow *select* en los *switches* de borde implicados, permitiendo repartir el tráfico entre dos ramas equivalentes.

La instalación se realizó directamente en Open vSwitch mediante `ovs-ofctl`. Esta decisión no debe interpretarse como la línea final SDN del proyecto, sino como una comprobación controlada de bajo nivel. Antes de intentar la instalación mediante OpenDaylight RESTCONF era necesario verificar que el *datapath* aceptaba los grupos, que el tráfico atravesaba correctamente la ruta y que los contadores de ambas ramas permitían observar reparto efectivo.

La Tabla 8.6 resume la validación inicial en el plano de datos, comparando el camino único de referencia con el demostrador ECMP limitado.

Tabla 8.6: Demostrador ECMP limitado en el *datapath*

Configuración	**Instalación**	**Tráfico**	**Throughput medio**	**Observación**
Single-path	10 *flows*	PASS	32,72 Gbps	Camino único usado como referencia de comparación.
ECMP limitado	2 grupos y 16 *flows*	PASS	32,08 Gbps	Ambas ramas equivalentes presentan contadores positivos.

La Figura 8.8 compara el *throughput* medio obtenido con camino único y con el demostrador ECMP limitado. Los valores son muy próximos: 32,72 Gbps en *single-path* y 32,08 Gbps en ECMP limitado. La ligera diferencia no debe interpretarse como una pérdida estructural del mecanismo *multipath*, sino como una variación esperable dentro del entorno emulado y de una prueba acotada. Lo relevante en esta etapa no es maximizar el *throughput*, sino demostrar que la instalación con grupo *select* mantiene tráfico válido y activa más de una rama.

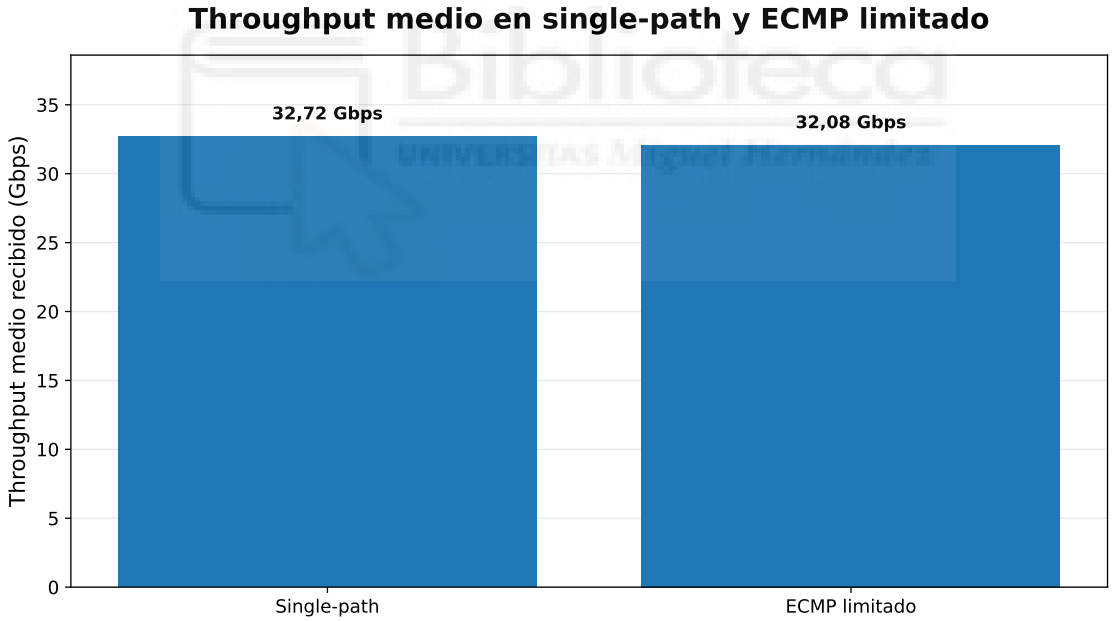

Figura 8.8: *Throughput* medio en *single-path* y ECMP limitado.

La Figura 8.9 muestra el reparto de tráfico entre las dos ramas equivalentes. En la configuración *single_path*, el tráfico se concentra en una única rama, como era esperable. En la configuración ECMP limitada, los contadores reflejan un reparto aproximado del 57,68 % y 42,32 %. Este resultado confirma que el grupo *select* no se limita a instalarse formalmente, sino que participa realmente en el plano de datos.

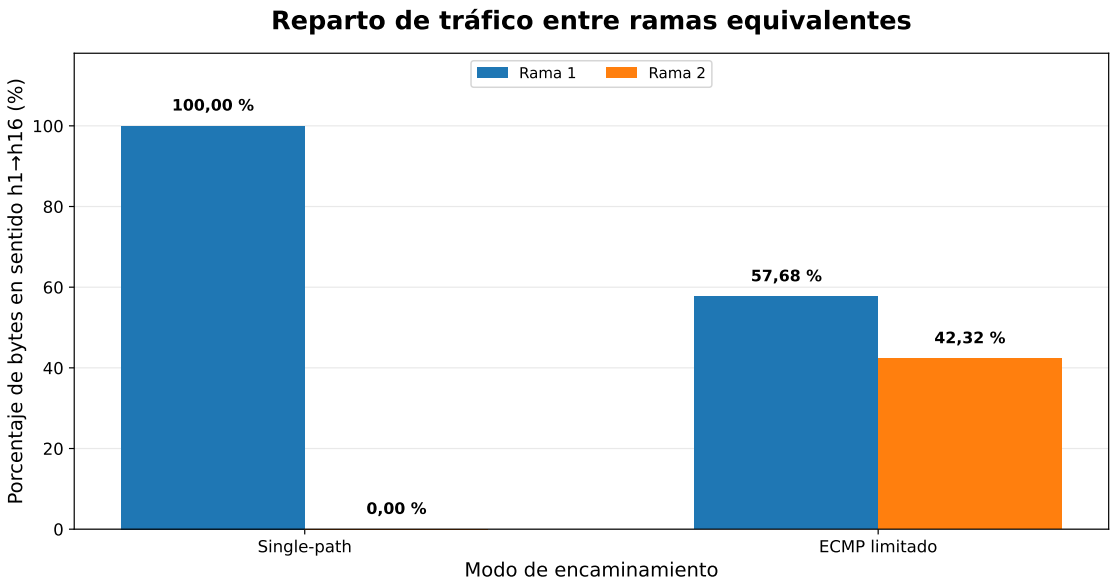

Figura 8.9: Reparto de tráfico en el demostrador ECMP limitado.

La conclusión de esta primera etapa es positiva, pero limitada. Se valida que el *datapath* OpenFlow puede ejecutar un demostrador *multipath* acotado mediante grupos *select*. Sin embargo, al haber sido instalado directamente con `ovs-ofctl`, este resultado no demuestra por sí solo que OpenDaylight haya programado el mecanismo. Por ello, la siguiente etapa consistió en trasladar esta misma lógica a RESTCONF.

8.11.2. Instalación del demostrador mediante OpenDaylight RESTCONF

La segunda etapa tuvo como objetivo comprobar si OpenDaylight Titanium podía instalar grupos OpenFlow *select* y los flujos asociados mediante RESTCONF sobre los *switches* de borde implicados en el demostrador. Para ello se trabajó con los *switches* $e1$ y $e8$, mapeados respectivamente a los nodos `openflow:13` y `openflow:20`. En ambos casos se instaló el grupo `group_id=100`, con dos *buckets* de salida hacia ramas equivalentes.

La Tabla 8.7 resume la instalación RESTCONF del demostrador. La instalación por RPC funcionó con la variante `json_input`, y la instalación en *datastore* funcionó con la variante `ns_group`. Esta información se conserva porque documenta la forma concreta en que OpenDaylight aceptó la definición del grupo.

Tabla 8.7: Instalación RESTCONF del demostrador ECMP limitado

Elemento	Switches	Método	Estado	Detalle
Grupo *select* en `e1`	`e1`	RPC RESTCONF de grupo	PASS	Variante RPC: `json_input` Datastore: `ns_group`
Grupo *select* en `e8`	`e8`	RPC RESTCONF de grupo	PASS	Variante RPC: `json_input` Datastore: `ns_group`

Flujos de encaminamiento ECMP limitado	`e1`, `e8`
	`a1`, `a2`
	`c1`, `c3`
	`a7`, `a8`
		RESTCONF datastore PUT	PASS	16/16 flujos instalados y verificados en Open vSwitch.

Tras instalar los grupos, se instalaron los flujos necesarios para encaminar el tráfico $h1 \leftrightarrow h16$. Esta segunda etapa cerró con 16 de 16 flujos verificados en Open vSwitch. La validación posterior confirmó que los grupos aparecían tanto en OVS como en el inventario operacional de OpenDaylight, y que el tráfico real generaba contadores positivos en las ramas.

La Tabla 8.8 resume las métricas principales del tráfico validado mediante RESTCONF: pérdida nula, RTT medio, *throughput*, evidencia de grupos y evidencia de ramas activas.

Tabla 8.8: Métricas del demostrador ECMP limitado RESTCONF

Métrica	Valor	Unidad	Observación
Pérdida en ping de control	0,00	%	Conectividad validada entre `h1` y `h16`.
RTT medio en ping de control	0,224	ms	Media reportada por `ping` durante la prueba funcional.
Throughput medio recibido	31,46	Gbps	Resultado medio obtenido con `iperf3`; 3 de 3 repeticiones válidas.
Evidencia de grupos	Sí	–	Grupo OpenFlow *select* presente y con contadores positivos.
Evidencia de ramas	Sí	–	Ambas ramas equivalentes aparecen activas durante la prueba.

El ensayo de tráfico $h1 \rightarrow h16$ finalizó con 0 % de pérdida en los pings de control, 3 de 3 repeticiones válidas en `iperf3` y un *throughput* medio de 31,46 Gbps. Además, las evidencias de grupo y de ramas fueron positivas, lo que confirma que el mecanismo no solo fue aceptado por RESTCONF, sino que quedó operativo en el *datapath*.

La Figura 8.10 recoge el *throughput* medio del tráfico validado mediante RESTCONF. El valor obtenido, 31,46 Gbps, se mantiene en el mismo orden que el observado en la validación inicial sobre el *datapath*, lo que indica que el traslado de la prueba a RESTCONF no introdujo una degradación severa en el entorno emulado.

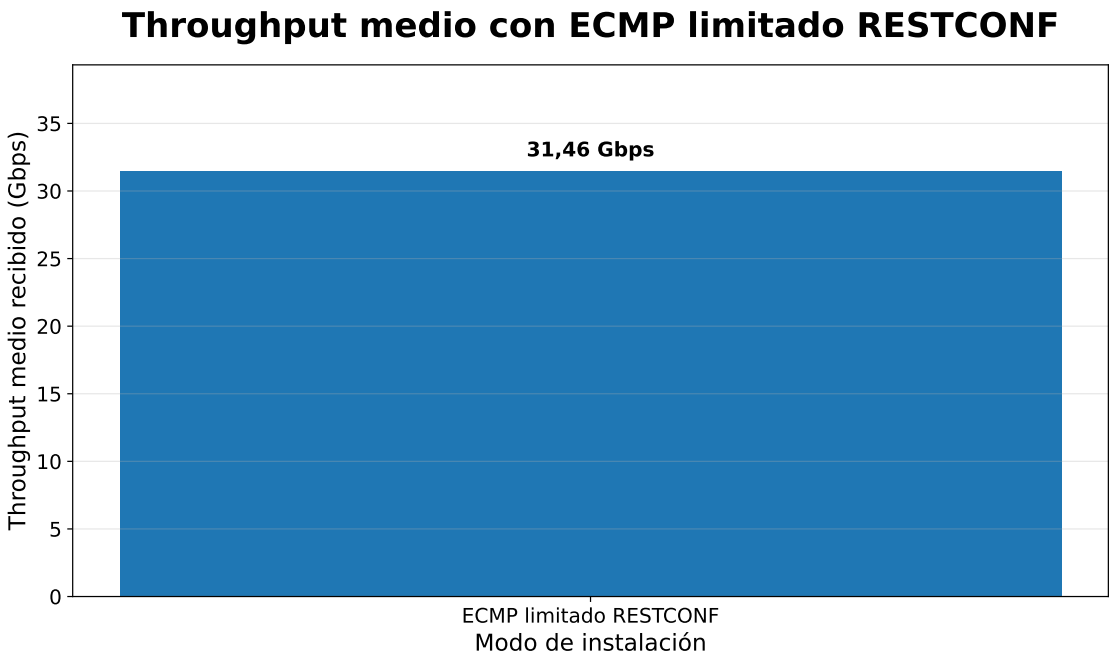

Figura 8.10: *Throughput* del demostrador ECMP limitado RESTCONF.

La Figura 8.11 muestra el reparto de tráfico observado en la instalación RESTCONF. En el sentido $h1 \rightarrow h16$, las ramas presentan un reparto aproximado del 42,15 % y 57,85 %. En el sentido inverso $h16 \rightarrow h1$, el reparto fue aproximadamente del 43,12 % y 56,88 %. Estos valores refuerzan la interpretación de que ambas ramas equivalentes fueron utilizadas de forma efectiva.

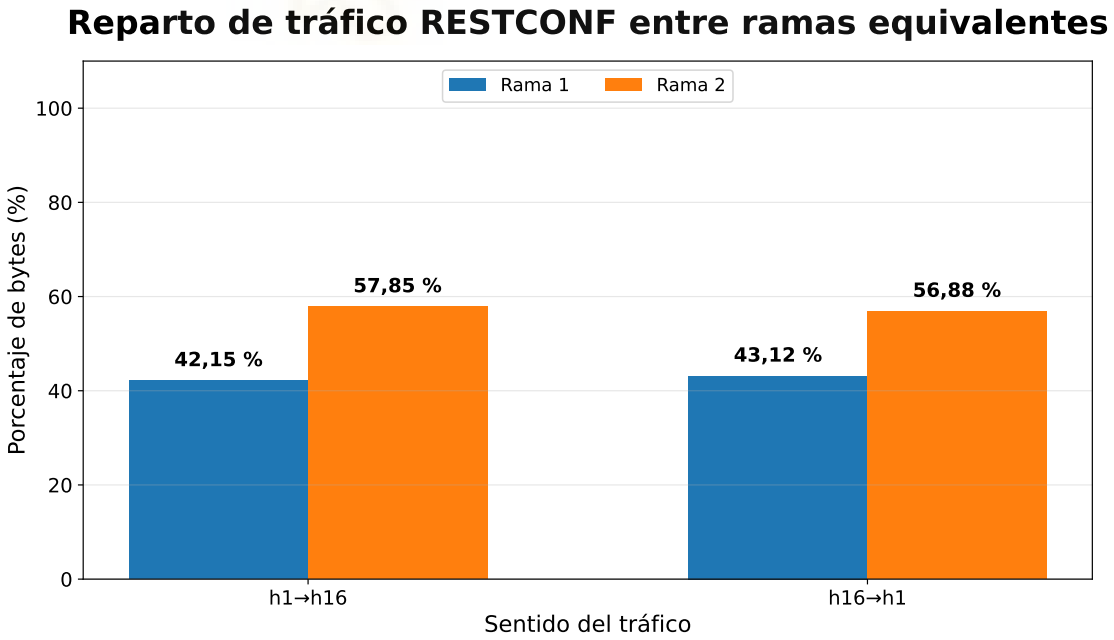

Figura 8.11: Reparto de tráfico RESTCONF entre ramas equivalentes.

La Figura 8.12 muestra el RTT medio observado en los pings de control del demostrador RESTCONF. Este resultado se utiliza como validación funcional de conectividad y no

como comparación estadística amplia de latencia, ya que la prueba se limita a un único par de *hosts*.

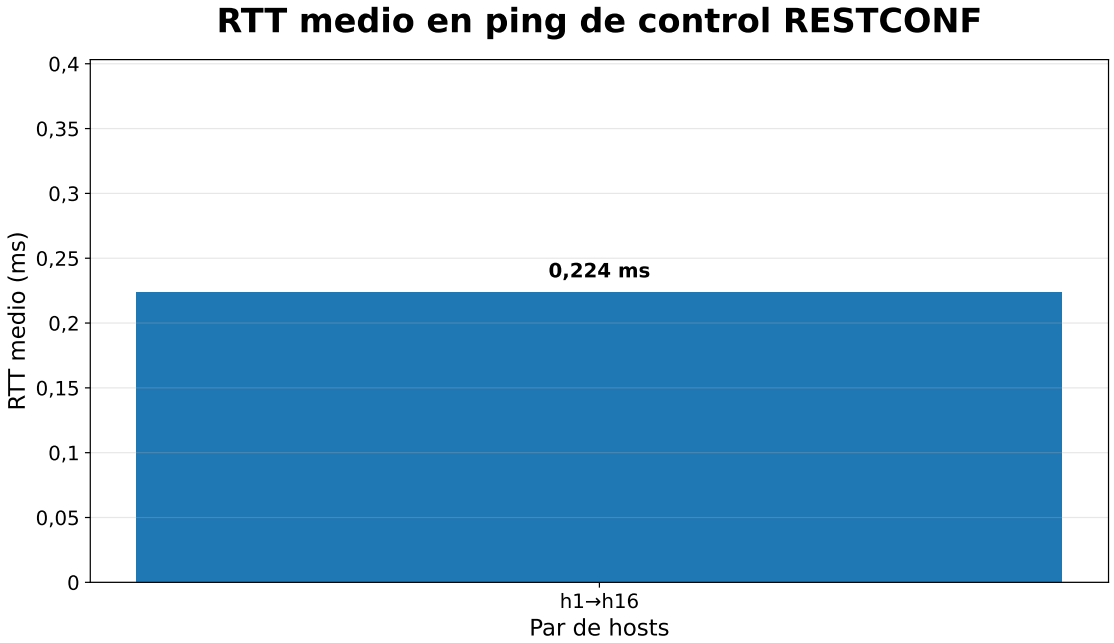

Figura 8.12: RTT del demostrador ECMP limitado RESTCONF.

El Código 8.8 muestra el resumen operativo de instalación de grupos mediante RESTCONF. Se conserva como evidencia porque documenta explícitamente la variante aceptada por OpenDaylight y la presencia del grupo en OVS.

```
python3 tfg_sdn/experiments/ch08_restconf_group_select_demo.py \
  --campaign-id ch08_fattree_ecmp_restconf_groups_20260520_1009 \
  --action install-groups-restconf

{
  "campaign_id": "ch08_fattree_ecmp_restconf_groups_20260520_1009",
  "status": "PASS",
  "scope": "RESTCONF RPC + datastore PUT installation for e1/e8 group-select"
}

Resultado relevante:
- Grupo select e1: PASS (RPC=json_input; DS=ns_group)
- Grupo select e8: PASS (RPC=json_input; DS=ns_group)
```

Código 8.8: Instalación RESTCONF de grupos OpenFlow *select*.

El Código 8.9 resume la instalación de flujos mediante RESTCONF *datastore*. La aceptación de 16 de 16 flujos y su verificación posterior en OVS constituyen la evidencia principal de que el demostrador no depende únicamente de reglas locales instaladas a mano.

```
python3 tfg_sdn/experiments/ch08_restconf_group_select_demo.py \
  --campaign-id ch08_fattree_ecmp_restconf_groups_20260520_1009 \
  --action install-flows-restconf

{
  "campaign_id": "ch08_fattree_ecmp_restconf_groups_20260520_1009",
  "method": "PUT /rests/data/.../flow=<flow_id>",
  "n_flows": 16,
  "n_ok": 16,
  "status": "PASS",
  "scope": "RESTCONF datastore PUT installation for ECMP-limited h1-h16 forwarding"
}
```

Código 8.9: Instalación RESTCONF de flujos del demostrador ECMP limitado.

Finalmente, el Código 8.10 recoge el resumen de tráfico del demostrador RESTCONF. Esta salida confirma conectividad, repeticiones válidas de `iperf3`, *throughput* medio y evidencias positivas de grupos y ramas.

```
ping_ok: True
packet_loss_pct: 0.0
rtt_avg_ms: 0.224
n_iperf_ok: 3/3
throughput_mean_gbps: 31.46
group_evidence_ok: True
branch_evidence_ok: True
forward_1_share_pct: 42.2
forward_2_share_pct: 57.8
reverse_1_share_pct: 43.1
reverse_2_share_pct: 56.9
status: PASS
```

Código 8.10: Resumen de tráfico del demostrador ECMP limitado RESTCONF.

8.11.3. Comparación metodológica entre ambas etapas

La comparación entre ambas etapas permite separar dos niveles de validación. La validación inicial demostró que el *datapath* podía ejecutar un grupo *select* y repartir tráfico entre ramas equivalentes cuando la programación se realizaba directamente con `ovs-ofctl`. La validación mediante RESTCONF añadió una capa adicional: la instalación de grupos y flujos desde OpenDaylight, manteniendo posteriormente la verificación contra Open vSwitch.

Desde el punto de vista de resultados, el *throughput* medio fue de 32,08 Gbps en el demostrador programado directamente en el *datapath* y de 31,46 Gbps en el demostrador instalado mediante RESTCONF. La diferencia es reducida y no modifica la conclusión principal. La aportación de la segunda etapa no es obtener más *throughput*, sino demostrar que el mecanismo puede instalarse desde la controladora y quedar reflejado en el plano de datos.

La Tabla 8.9 resume esta comparación metodológica, diferenciando el mecanismo de instalación, el estado de la prueba, el *throughput* medio y la interpretación de cada etapa.

Tabla 8.9: Comparación metodológica del demostrador ECMP limitado

Etapa	Mecanismo	Estado	Throughput medio	Interpretación
Validación en *datapath*	Programación directa con `ovs-ofctl`	PASS	32,08 Gbps	Valida la viabilidad del *datapath* y el uso de dos ramas equivalentes.
Validación mediante RESTCONF	Instalación mediante OpenDaylight RESTCONF	PASS	31,46 Gbps	Valida la instalación controlada del demostrador desde la controladora SDN.

En consecuencia, esta extensión debe interpretarse como una evidencia adicional y no como el resultado principal del capítulo. El bloque principal sigue siendo la campaña RESTCONF determinista, porque es la campaña sistemática sobre varios escenarios de tráfico. El demostrador ECMP limitado añade una evidencia exploratoria: muestra que el uso de grupos OpenFlow *select* es viable sobre una ruta representativa de la Fat-Tree y que esta lógica puede instalarse mediante RESTCONF.

8.11.4. Limitaciones específicas del demostrador ECMP limitado

El demostrador presenta varias limitaciones que deben quedar explícitas para evitar una sobreinterpretación de los resultados. En primer lugar, se limita al par $h1 \leftrightarrow h16$. Por tanto, no valida ECMP para todos los pares de *hosts* ni para toda la matriz de tráfico de la Fat-Tree.

En segundo lugar, el mecanismo no implementa balanceo dinámico ni reconfiguración adaptativa. Los grupos y flujos se instalan antes de la prueba, y el sistema no modifica los caminos en función de congestión, utilización de enlaces o telemetría en tiempo real.

En tercer lugar, aunque se observan dos ramas activas, el reparto exacto entre ellas no debe interpretarse como una garantía matemática de balanceo perfecto. El comportamiento de los grupos *select* depende de la implementación del *datapath*, de los hashes internos y del patrón de tráfico utilizado.

En cuarto lugar, el uso de Mininet implica que los valores absolutos de *throughput* dependen de los recursos compartidos de la máquina anfitriona. Por tanto, las cifras deben interpretarse como resultados internos del banco de pruebas y no como valores extrapolables directamente a una infraestructura física.

Estas limitaciones no invalidan el demostrador. Al contrario, delimitan correctamente lo demostrado: un caso ECMP limitado, verificable y trazable, primero en el *datapath* y posteriormente instalado mediante OpenDaylight RESTCONF.

8.11.5. Conclusión de la extensión ECMP limitada

La extensión ECMP limitada confirma que el banco de pruebas desarrollado no solo permite instalar *forwarding* determinista sobre Fat-Tree k=4, sino también explorar mecanismos *multipath* mediante grupos OpenFlow *select*. La primera validación confirmó la viabilidad del *datapath* mediante programación directa en OVS, mientras que la segunda demostró que la instalación de grupos y flujos puede realizarse mediante OpenDaylight RESTCONF para un caso acotado.

El resultado más relevante no es una mejora de *throughput* frente al camino único, sino la activación verificable de ramas equivalentes y la conservación de evidencias en forma de contadores, resúmenes JSON, tablas, figuras y listados. Desde el punto de vista académico, esta extensión refuerza la continuidad metodológica entre el bloque Leaf-Spine y la topología Fat-Tree. Leaf-Spine permitió introducir el concepto *multipath* en una arquitectura más simple, mientras que este demostrador muestra que dicho concepto puede reproducirse de forma limitada, controlada y verificable en Fat-Tree. Esta continuidad no implica que los valores de rendimiento de ambos bloques sean comparables de forma absoluta, ni que el éxito en Leaf-Spine demuestre automáticamente ECMP completo en Fat-Tree.

No obstante, el trabajo no debe presentar esta extensión como ECMP completo Fat-Tree. La implementación completa y generalizada de ECMP para todos los pares de *hosts*, con selección sistemática de caminos y posible reconfiguración dinámica basada en telemetría, queda fuera del alcance cerrado de esta campaña. Por este motivo, la sección siguiente no se plantea como una validación completa de ECMP en Fat-Tree, sino como un intento exploratorio adicional para comprobar hasta dónde podía extenderse el mecanismo *multipath* dentro del banco de pruebas.

8.12. Intento exploratorio hacia ECMP completo en Fat-Tree k=4

Tras cerrar el demostrador ECMP limitado, se realizó una última campaña exploratoria con el objetivo de estudiar si el mecanismo *multipath* podía extenderse hacia un modelo más ambicioso dentro de la topología Fat-Tree k=4. Esta campaña no sustituye a la evaluación principal del capítulo, basada en *forwarding* determinista instalado mediante OpenDaylight RESTCONF, ni convierte el demostrador anterior en una validación completa de ECMP. Su función es complementaria: documentar hasta dónde puede llevarse el banco de pruebas cuando se intenta activar selección de caminos en varios niveles de la jerarquía Fat-Tree.

La decisión de incluir esta campaña responde a un criterio metodológico conservador. En capítulos anteriores se validó que la redundancia estructural puede explotarse en topologías más simples, especialmente en Leaf-Spine. En Fat-Tree, sin embargo, la presencia de capas

de borde, agregación y núcleo hace que el problema sea más exigente. Por ello, en lugar de asumir que el mecanismo ECMP completo podía trasladarse directamente, se planteó una prueba acotada, con criterios de parada claros y con interpretación estricta de las evidencias obtenidas.

El objetivo inicial fue estudiar el par inter-*pod* `h1` ↔ `h16`, ya utilizado en fases anteriores, e incorporar posteriormente un segundo par de estrés `h2` ↔ `h15`. Para el sentido `h1` → `h16`, las rutas candidatas consideradas atraviesan los *switches* `e1`, `a1/a2`, `c1/c2/c3/c4`, `a7/a8` y `e8`. De forma simétrica, el sentido inverso `h16` → `h1` utiliza la misma estructura jerárquica en dirección opuesta. La hipótesis experimental era que los grupos *select* podrían inducir reparto de tráfico tanto en los *switches* de borde como en los *switches* de agregación.

La Tabla 8.10 resume el desarrollo de la campaña exploratoria, diferenciando el bloque de diseño e inventario, el prototipo *datapath*, la prueba de estrés y la migración RESTCONF.

Tabla 8.10: Resumen del intento exploratorio hacia ECMP completo

Bloque	**Estado**	**Evidencia principal**	**Interpretación**
Diseño e inventario	Completado	20 *bridges* OVS, 20 nodos OpenFlow y mapeo correcto.	Permite pasar al prototipo sin ambigüedad de inventario.
Prototipo *datapath*	Completado como prototipo exploratorio	Tráfico entre `h1` y `h16` con grupos *select*.	Se observa *multipath* en switches de borde.
Prueba de estrés *datapath*	Parcial en prueba de estrés	Dos pares de hosts y 16 ejecuciones `iperf3`.	No activa ambos *buckets* en agregación.
Migración RESTCONF	Parcial en prueba de estrés RESTCONF	Grupos y reglas instalados desde OpenDaylight.	Migración lograda, pero con resultado parcial.

La primera etapa consistió en capturar el estado de diseño e inventario del banco de pruebas, confirmando que la topología Fat-Tree k=4 estaba desplegada correctamente, que los 20 *bridges* Open vSwitch se encontraban conectados a OpenDaylight y que el inventario operacional mostraba los 20 nodos OpenFlow esperados. Esta comprobación permitió pasar al prototipo de *datapath* sin introducir ambigüedad en el mapeo entre *switches* OVS, identificadores DPID y nodos `openflow:<id>`.

El Código 8.11 recoge la secuencia reproducible empleada para la parte inicial de la campaña exploratoria. A diferencia de las primeras pruebas manuales, esta versión queda integrada en un flujo *Python-first*, coherente con el resto del proyecto.

```
# Secuencia Python-first del prototipo datapath
export CAMPAIGN_ID=ch08_fattree_ecmp_full_20260521_0853

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d0-capture

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d1-install

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d1-prepare-traffic

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d1-summarize

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d2-install-extra

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d2-prepare-stress

python3 tfg_sdn/experiments/ch08_ecmp_full_datapath_demo.py \
  --campaign-id "$CAMPAIGN_ID" --action d2-summarize
```

Código 8.11: Secuencia reproducible del prototipo *datapath*.

El prototipo *datapath* se instaló directamente con `ovs-ofctl`, no como solución final SDN, sino como paso intermedio para comprobar si el plano de datos era capaz de ejecutar la lógica de grupos *select* en varios *switches* de la ruta.

La Tabla 8.11 resume las métricas obtenidas en los cuatro escenarios procesados: prototipo *datapath* punto a punto, prueba de estrés *datapath*, migración RESTCONF punto a punto y prueba de estrés RESTCONF.

Tabla 8.11: Métricas de tráfico del intento exploratorio

Escenario	Método	Estado	Pérdida	RTT	iperf	Throughput	Buckets
Prototipo punto a punto	`ovs-ofctl`	Prototipo exploratorio	0,0 %	0,229 ms	4/4	70,80 Gbps	`e1`, `e8`
Estrés con dos pares	`ovs-ofctl`	Parcial en estrés	0,0 %	0,304 ms	16/16	275,95 Gbps	`e1`, `e8`
RESTCONF punto a punto	OpenDaylight RESTCONF	Parcial con tráfico	0,0 %	0,229 ms	4/4	70,28 Gbps	`e1`, `e8`
RESTCONF con estrés	OpenDaylight RESTCONF	Parcial en estrés	0,0 %	0,230 ms	16/16	282,04 Gbps	`e1`, `e8`

En todos los casos se obtuvo conectividad válida, sin pérdida de paquetes en los pings de control, y las ejecuciones de `iperf3` finalizaron correctamente. Los valores de *throughput* agregado fueron aproximadamente 70,80 Gbps en el prototipo *datapath* punto a punto, 275,95 Gbps en la prueba de estrés *datapath*, 70,28 Gbps en la migración RESTCONF punto a punto y 282,04 Gbps en la prueba de estrés RESTCONF.

Estos valores se emplean como evidencia interna de estabilidad y transporte de tráfico dentro de cada escenario procesado. En particular, los escenarios de estrés agregan varias ejecuciones y flujos concurrentes, por lo que no deben interpretarse como capacidad física instantánea de la topología ni compararse de forma directa con las campañas anteriores. La conclusión relevante no es el valor absoluto de *throughput*, sino que la instalación

exploratoria mantiene conectividad, tráfico válido y evidencias de contadores bajo una carga más exigente.

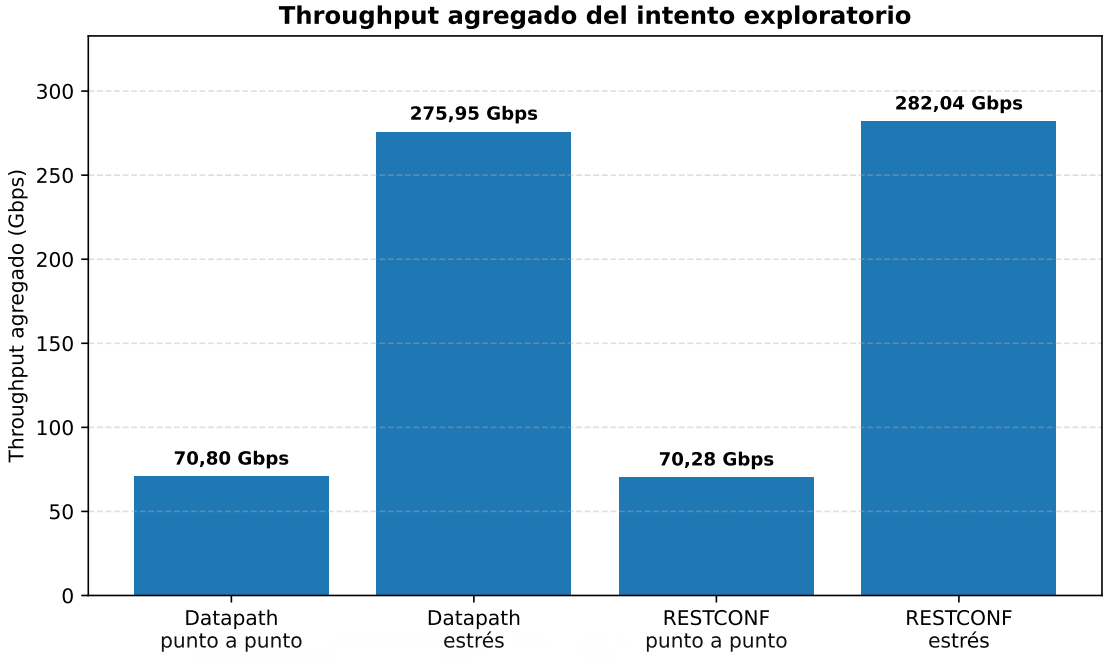

Figura 8.13: *Throughput* agregado del intento exploratorio.

La Figura 8.13 muestra que la migración a RESTCONF no degradó de forma relevante el comportamiento observado en el prototipo *datapath*. Los escenarios punto a punto se mantienen en torno a 70 Gbps, mientras que los escenarios de estrés alcanzan valores agregados superiores al sumar varias ejecuciones concurrentes. Esta figura no debe interpretarse como una comparación de capacidad absoluta de la topología, sino como evidencia de que el mecanismo instalado permite tráfico estable tanto en la versión *datapath* como en la versión migrada a RESTCONF.

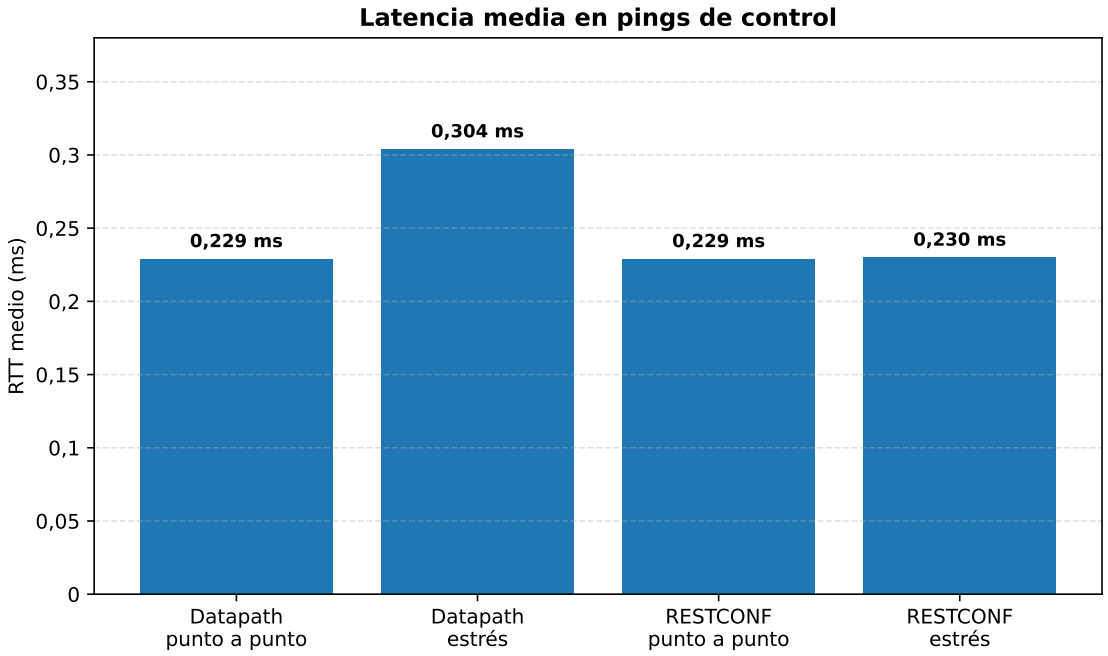

Figura 8.14: Latencia media en el intento exploratorio.

La Figura 8.14 complementa los resultados de *throughput* con la latencia media de control. Los valores se mantienen en el orden de décimas de milisegundo, coherentes con un entorno emulado local en Mininet. Esta estabilidad es importante porque confirma que las pruebas de rendimiento no se apoyan en un estado de conectividad inestable o parcialmente roto.

Una vez validado el prototipo *datapath*, se intentó migrar la misma lógica a OpenDaylight RESTCONF. Esta etapa es la más relevante desde el punto de vista SDN, porque evita que el resultado quede limitado a programación directa del *datapath* mediante `ovs-ofctl`. La migración consistió en instalar grupos OpenFlow *select* y reglas de encaminamiento mediante llamadas RESTCONF, verificar su presencia tanto en Open vSwitch como en el inventario operacional de OpenDaylight y repetir posteriormente las pruebas de tráfico.

El Código 8.12 resume la secuencia RESTCONF utilizada en la etapa final de la campaña exploratoria.

```
# Secuencia Python-first de la migracion RESTCONF
export CAMPAIGN_ID=ch08_fattree_ecmp_full_restconf_20260521_1705

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-cleanup-ovs

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-install-groups-restconf

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-verify-groups

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-install-flows-restconf

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-summarize-traffic

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-install-stress-flows-restconf

python3 tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py \
  --campaign-id "$CAMPAIGN_ID" --action d3-summarize-stress
```

Código 8.12: Secuencia reproducible de la migración RESTCONF.

Primero se limpió el estado previo del *datapath*, después se instalaron los grupos *select*, se verificó su presencia operacional, se instalaron los flujos punto a punto y, finalmente, se añadió la carga de estrés. Esta secuencia permite defender que la campaña no se limita a una inspección manual, sino que conserva una trazabilidad clara entre instalación, verificación y generación de tráfico.

La Tabla 8.12 confirma que la migración RESTCONF se completó correctamente en términos de instalación. Los grupos *select* fueron instalados y verificados en los *switches* implicados, y las reglas de encaminamiento se insertaron mediante *datastore PUT*.

Tabla 8.12: Instalación RESTCONF del intento exploratorio

Elemento	Estado	Resultado	Observación
Limpieza de *datapath*	Completado	`e1`, `e8`, `a1`, `a2`, `a7`, `a8` y switches *core*.	Preparación limpia antes de instalar grupos y reglas.
Instalación de grupos *select*	Completado	6/6 switches con grupo *select*.	RPC correcto y persistencia en *datastore*.
Verificación operacional de grupos	Completado	Presencia en OVS y en inventario operacional de OpenDaylight.	La instalación queda confirmada en controladora y *datapath*.
Instalación de reglas punto a punto	Completado	20/20 reglas instaladas.	Instalación mediante RESTCONF *datastore* PUT.
Instalación de reglas de estrés	Completado	20/20 reglas instaladas.	Preparación de la prueba con tráfico adicional.

Esta evidencia permite afirmar que OpenDaylight fue capaz de programar el mecanismo exploratorio en el plano de datos. Sin embargo, la presencia de grupos y reglas no es

suficiente para afirmar ECMP completo; es necesario analizar los contadores por *bucket* tras generar tráfico real.

El Código 8.13 sintetiza el modelo de instalación empleado.

```
Modelo de instalacion RESTCONF empleado
=======================================
Grupos select instalados en: e1, a1, a2, e8, a7, a8
Identificador de grupo: 200
Reglas con bifurcacion: destino Ethernet del host remoto y accion hacia grupo select
Reglas deterministas: destino Ethernet del host remoto y salida por puerto concreto
Verificacion en OVS: descripcion de grupos y contadores por bucket
Verificacion en OpenDaylight: inventario operacional con presencia del grupo select
```

Código 8.13: Modelo de instalación RESTCONF empleado en el intento exploratorio.

Los grupos *select* se instalaron en los *switches* `e1`, `a1`, `a2`, `e8`, `a7` y `a8`, con reglas que enviaban determinados destinos Ethernet hacia el grupo correspondiente y otras reglas deterministas que completaban la trayectoria hacia el *host* final. Esta combinación permitió construir un prototipo jerárquico más rico que el demostrador ECMP limitado, aunque todavía acotado a pares de *hosts* concretos.

La Tabla 8.13 recoge la evidencia principal de contadores. En los *switches* de borde `e1` y `e8` se activaron ambos *buckets* del grupo *select*, lo que confirma bifurcación real del tráfico en el inicio de cada sentido. En cambio, los *switches* de agregación `a1`, `a2`, `a7` y `a8` presentaron contadores positivos de grupo, pero con actividad concentrada en un único *bucket*.

Tabla 8.13: Evidencia de grupos OpenFlow *select*

Switch	Rol	Paquetes	Tráfico	Buckets	Bucket 0	Bucket 1
e1	Borde origen	6,43 M	282,21 GB	2/2	31,5 %	68,5 %
a1	Agregación	2,02 M	88,76 GB	1/2	100,0 %	0,0 %
a2	Agregación	4,40 M	193,45 GB	1/2	0,0 %	100,0 %
e8	Borde destino	5,77 M	0,38 GB	2/2	30,8 %	69,2 %
a7	Agregación	1,78 M	0,12 GB	1/2	100,0 %	0,0 %
a8	Agregación	3,99 M	0,26 GB	1/2	0,0 %	100,0 %

Este resultado es el punto clave de la interpretación: el prototipo funcionó, fue migrado a RESTCONF y transportó tráfico correctamente, pero no logró demostrar activación de varios *buckets* en la capa de agregación. Las Figuras 8.15 y 8.16 sintetizan esta limitación desde dos perspectivas complementarias: la activación de *buckets* por *switch* y la distribución porcentual del tráfico por *bucket*.

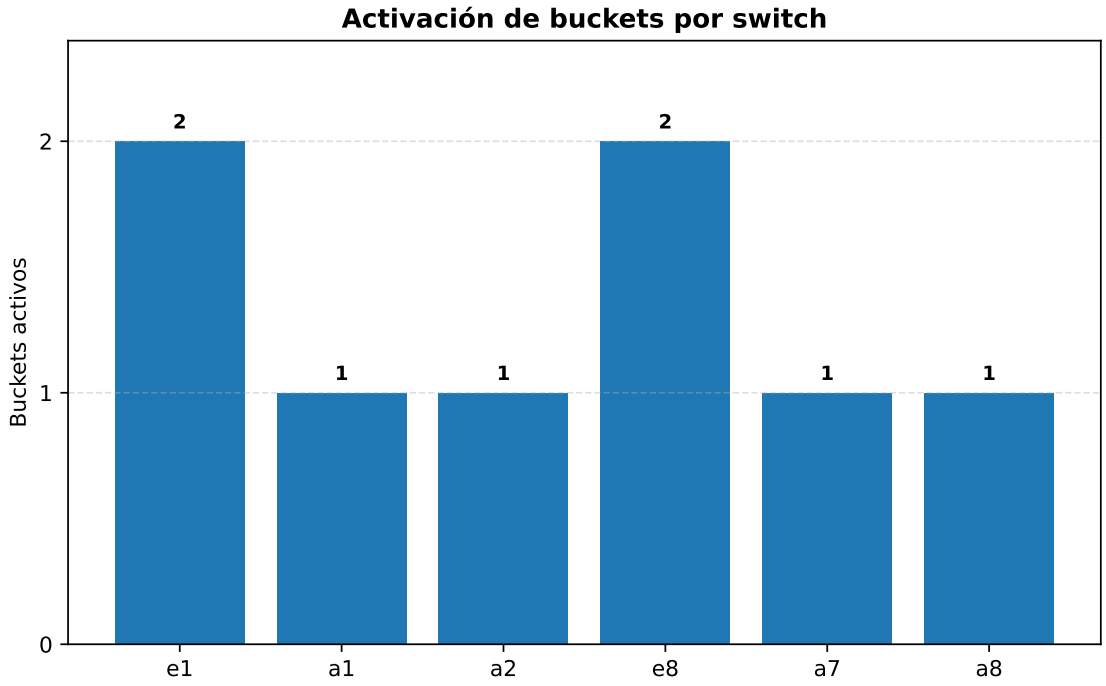

Figura 8.15: Activación de *buckets* por *switch*.

En la activación por *switch* se observa que `e1` y `e8` alcanzan dos *buckets* activos, mientras que `a1`, `a2`, `a7` y `a8` permanecen con un único *bucket* activo. Por tanto, la campaña no permite afirmar ECMP completo en toda la Fat-Tree. Sí permite, en cambio, documentar un intento avanzado en el que se instalaron grupos en varias capas y se verificó actividad real en todos los *switches* implicados.

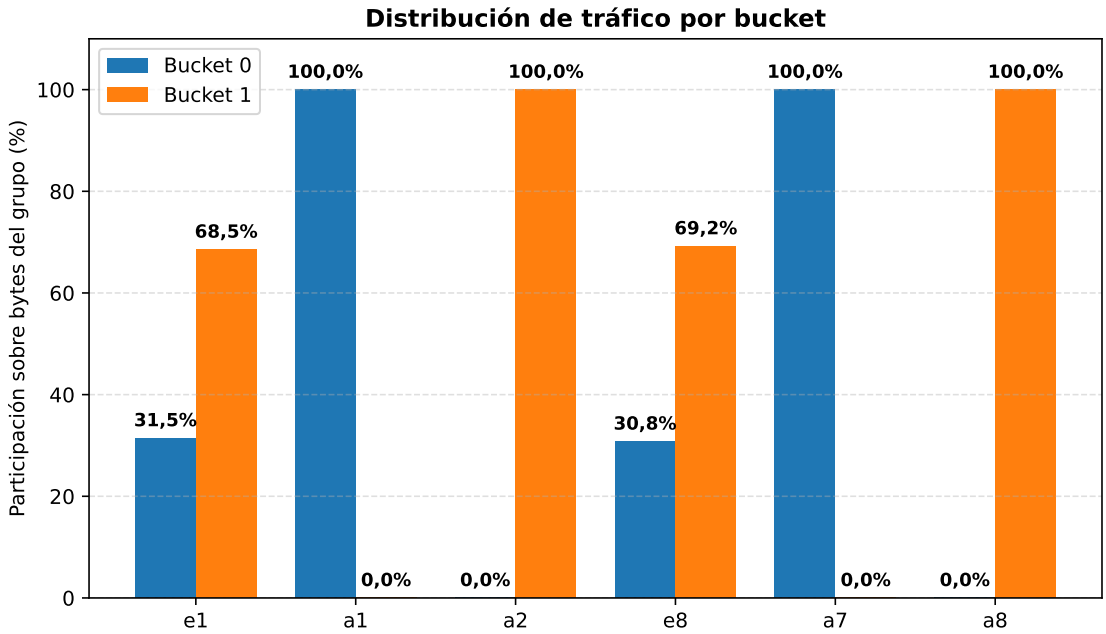

Figura 8.16: Distribución de tráfico por *bucket*.

La Figura 8.16 refuerza la misma conclusión desde el punto de vista porcentual. En `e1` y `e8` aparece reparto entre ambos *buckets*, aunque no necesariamente equilibrado al 50 %. En los *switches* de agregación, por el contrario, el 100 % del tráfico de grupo se concentra en un único *bucket*. Esta polarización puede explicarse por la combinación de reglas deterministas, granularidad del tráfico utilizado, comportamiento de selección de Open vSwitch y alcance reducido del escenario. Lo importante es que el resultado no se sobreinterpreta: hay evidencia de *multipath* en borde, pero no de ECMP completo en toda la jerarquía.

La Tabla 8.14 resume la interpretación defendible de la campaña exploratoria. La migración a OpenDaylight RESTCONF se consiguió, el tráfico fue estable y se obtuvo activación de varios *buckets* en los *switches* de borde.

Tabla 8.14: Interpretación del intento exploratorio hacia ECMP completo

Aspecto	**Resultado**	**Base experimental**	**Conclusión**
Migración a OpenDaylight RESTCONF	Sí	Grupos y reglas instalados mediante RESTCONF.	Queda validada la trazabilidad entre OpenDaylight y Open vSwitch.
Tráfico estable	Sí	Pings sin pérdida y ejecuciones de `iperf3` correctas.	La campaña es reproducible dentro del banco de pruebas.
Multipath en borde	Sí	`e1` y `e8` activan dos *buckets*.	Existe evidencia clara de bifurcación en los switches de borde.
Multipath en agregación	No	`a1`, `a2`, `a7` y `a8` usan un solo *bucket*.	No se afirma ECMP completo en toda la jerarquía.
ECMP completo Fat-Tree	No	Falta activación de varios *buckets* en agregación y *core*.	El resultado debe interpretarse como parcial y exploratorio.

Sin embargo, al no observarse activación de varios *buckets* en agregación, el resultado debe describirse como parcial y exploratorio. Esta conclusión es más sólida que presentar el intento exploratorio como un fracaso, porque el banco de pruebas sí demostró capacidad para instalar grupos y flujos mediante RESTCONF, verificación operacional y transporte de tráfico bajo un modelo *multipath* acotado. Al mismo tiempo, evita afirmar un resultado que los contadores no respaldan.

El Código 8.14 recoge el estado final de la campaña exploratoria.

```
Estado final del intento exploratorio
====================================
Diseno e inventario: completado
Prototipo datapath: completado como prototipo exploratorio
Prueba de estres datapath: resultado parcial en prueba de estres datapath
Instalacion de grupos RESTCONF: completado
Verificacion de grupos RESTCONF: completado
Instalacion de reglas RESTCONF: completado (20/20)
Trafico RESTCONF: resultado parcial con trafico RESTCONF (4/4)
Reglas de estres RESTCONF: completado (20/20)
Prueba de estres RESTCONF: resultado parcial en prueba de estres RESTCONF (16/16)
Switches con dos buckets activos: e1, e8
Switches de agregacion con dos buckets activos: ninguno
```

Código 8.14: Estado final del intento exploratorio hacia ECMP completo.

En conjunto, este bloque cumple su objetivo metodológico: intentar extender el mecanismo *multipath* de forma controlada, obtener evidencias reproducibles y delimitar con precisión qué parte del mecanismo funciona y qué parte queda fuera del alcance validado. La conclusión final es que la topología Fat-Tree k=4 admite un prototipo *multipath* instalado mediante RESTCONF con actividad real en los *switches* de borde, pero no se dispone de evidencia suficiente para afirmar ECMP completo a través de borde, agregación y núcleo.

Desde el punto de vista del proyecto, esta campaña exploratoria tiene valor como cierre experimental. La campaña principal del capítulo sigue siendo el *forwarding* determinista mediante OpenDaylight RESTCONF, por ser la evaluación más estable, completa y trazable. El demostrador ECMP limitado confirma que los grupos OpenFlow *select* pueden utilizarse de forma acotada en Fat-Tree. La campaña exploratoria final añade un intento más ambicioso y documenta honestamente su límite: la migración RESTCONF fue viable, pero el reparto completo en la jerarquía Fat-Tree no quedó demostrado. Por ello, el ECMP completo se mantiene como línea de trabajo futuro, no como resultado cerrado del TFG.

En síntesis, el Capítulo 8 debe leerse como una secuencia de evidencias con distinto alcance. La campaña RESTCONF determinista constituye el resultado principal; el demostrador ECMP limitado aporta una validación acotada del uso de grupos *select*; y el intento exploratorio hacia ECMP completo documenta una migración RESTCONF viable pero parcial. Esta jerarquía de evidencias evita sobreinterpretar los resultados y mantiene la coherencia metodológica con las limitaciones formuladas en el Capítulo 7.

9. DISCUSIÓN

El capítulo anterior ha presentado la evaluación experimental de la topología Fat-Tree k=4 implementada en el banco de pruebas SDN. La campaña principal se basó en la instalación de reglas de *forwarding* determinista mediante OpenDaylight RESTCONF, mientras que los bloques posteriores exploraron el uso de grupos OpenFlow de tipo *select* para introducir mecanismos *multipath* de complejidad creciente. Este capítulo discute de forma conjunta esos resultados, poniendo el foco en su alcance real, sus implicaciones técnicas y las limitaciones que deben tenerse en cuenta antes de extraer conclusiones generales.

La discusión no introduce nuevos experimentos, sino que interpreta las evidencias ya obtenidas. En particular, se distingue entre tres niveles de resultado: la campaña RESTCONF determinista como evidencia principal del funcionamiento del banco de pruebas Fat-Tree, el demostrador ECMP limitado como extensión técnica validada en un escenario acotado, y el intento exploratorio hacia ECMP completo como evidencia parcial de viabilidad, pero no como una implementación completa de ECMP en toda la jerarquía Fat-Tree. Esta separación es esencial para mantener la trazabilidad metodológica del proyecto y evitar atribuir a la arquitectura evaluada propiedades que no hayan sido demostradas experimentalmente.

También debe mantenerse separada la lectura metodológica de los distintos bloques experimentales del TFG. Los resultados Leaf-Spine del Capítulo 5 se diseñaron con enlaces ascendentes limitados a 100 Mbps para observar saturación controlada y validar la intuición *multipath* en una arquitectura reducida. Los resultados Fat-Tree del Capítulo 8, en cambio, responden a una campaña distinta, centrada en validar una arquitectura jerárquica mediante *forwarding* instalado desde OpenDaylight RESTCONF. Por tanto, la discusión no compara ambas topologías en términos de capacidad absoluta, sino en términos de progresión metodológica, complejidad estructural y alcance de las evidencias obtenidas.

9.1. Síntesis global de los resultados

La Tabla 9.1 resume los bloques experimentales más relevantes del Capítulo 8: la campaña RESTCONF determinista, el demostrador ECMP limitado, el intento exploratorio hacia ECMP completo y la lectura global del conjunto de resultados.

Tabla 9.1: Síntesis global de resultados

Bloque	**Evidencia principal**	**Resultado defendible**	**Alcance**
RESTCONF determinista	Instalación y verificación de reglas mediante OpenDaylight RESTCONF, conectividad funcional y tráfico en escenarios punto a punto, concurrentes y *hotspot*.	Constituye la campaña principal del Capítulo 8.	Valida el banco de pruebas Fat-Tree k=4 con *forwarding* explícito, trazable y reproducible.
ECMP limitado	Uso de grupos OpenFlow *select* en un escenario acotado entre `h1` y `h16`.	Demuestra activación de ramas equivalentes en un caso controlado.	No representa ECMP completo ni conectividad universal entre todos los pares de hosts.
Intento hacia ECMP completo	Instalación de grupos y flujos en varios niveles de la jerarquía, con migración a RESTCONF y tráfico estable.	Resultado parcial: *multipath* observable en borde, pero no en agregación.	Sirve como exploración técnica y delimita el trabajo futuro.
Lectura global	Progresión desde validación determinista hasta exploración *multipath*.	El proyecto es defendible por su trazabilidad y honestidad metodológica.	No debe presentarse como una implementación final de ECMP completo Fat-Tree.

Desde el punto de vista de la defendibilidad académica, el resultado principal del proyecto no es la obtención de ECMP completo, sino la implementación de una arquitectura Fat-Tree controlada mediante SDN y evaluada con una metodología reproducible. La campaña determinista instalada desde OpenDaylight RESTCONF permite demostrar que el banco de pruebas puede desplegar reglas explícitas, verificar su presencia en el *datapath* y ejecutar escenarios de tráfico representativos. Esta base experimental es la que sostiene el Capítulo 8.

Las extensiones ECMP cumplen un papel diferente. No sustituyen a la campaña principal, sino que la amplían desde una perspectiva exploratoria. El demostrador ECMP limitado muestra que Open vSwitch y OpenDaylight pueden utilizar grupos *select* para repartir tráfico entre ramas equivalentes en un escenario controlado. En cambio, el intento exploratorio hacia ECMP completo permite identificar las dificultades que aparecen al extender esa lógica a varios niveles de la jerarquía Fat-Tree.

Esta jerarquía de resultados evita comparar bloques con distinto alcance como si fueran equivalentes. La campaña RESTCONF determinista aporta la evidencia principal de funcionamiento de Fat-Tree; el ECMP limitado aporta una validación acotada del mecanismo *select*; y el intento hacia ECMP completo aporta una frontera técnica documentada. Del mismo modo, los resultados Leaf-Spine deben entenderse como antecedente metodológico y no como una campaña de rendimiento directamente comparable con Fat-Tree.

9.2. Comparación crítica de las estrategias de *forwarding*

La Figura 9.1 sintetiza la lectura crítica de las estrategias de *forwarding* evaluadas. La figura no sustituye a las tablas de resultados, sino que ofrece una interpretación visual del incremento de ambición técnica a lo largo del proyecto: cuanto más se intenta explotar la redundancia estructural del *fabric*, mayor es la complejidad de control y mayor debe ser también la exigencia de validación.

Figura 9.1: Síntesis crítica de las estrategias de *forwarding*.

La Tabla 9.2 compara las tres estrategias desde el punto de vista de sus ventajas, limitaciones e interpretación dentro del TFG. Esta comparación es importante porque no todas las evidencias tienen el mismo peso. La campaña determinista presenta la mayor cobertura experimental, mientras que los bloques ECMP aportan valor técnico adicional, pero con un alcance más restringido.

Tabla 9.2: Comparación de estrategias de *forwarding*

Estrategia	Ventaja principal	Limitación principal	Interpretación en el TFG
Forwarding determinista	Rutas explícitas, controladas y verificables mediante RESTCONF y Open vSwitch.	No explota automáticamente la redundancia disponible.	Es la base experimental principal por su estabilidad, trazabilidad y cobertura.
ECMP limitado	Introduce grupos *select* y reparto entre ramas equivalentes en un caso acotado.	Se limita a un par de hosts y a una estructura de caminos concreta.	Aporta evidencia adicional de viabilidad *multipath*, sin sustituir a la campaña determinista.
Intento hacia ECMP completo	Explora instalación de grupos y flujos en varios niveles de la Fat-Tree.	No consigue activar varios *buckets* en agregación de forma suficiente.	Debe interpretarse como resultado parcial, exploratorio y útil para trabajo futuro.

La estrategia determinista tiene una ventaja metodológica clara: cada ruta instalada es explícita y verificable. Esto facilita relacionar cada resultado de tráfico con un conjunto concreto de reglas OpenFlow. Su limitación principal es que no explota automáticamente la redundancia disponible en la topología. Sin embargo, dentro del alcance del TFG, esa aparente limitación se convierte también en una ventaja: permite construir una campaña principal estable, trazable y fácil de defender.

El ECMP limitado introduce una capa adicional de complejidad al utilizar grupos *select*, pero conserva un escenario suficientemente acotado para que la interpretación sea robusta. En este caso, la activación de ramas equivalentes puede validarse mediante contadores de grupo y de *bucket*, sin necesidad de afirmar que toda la topología Fat-Tree está realizando ECMP completo.

El intento hacia ECMP completo es el bloque técnicamente más ambicioso, pero también el más delicado. La instalación de grupos y flujos mediante RESTCONF fue viable, y el tráfico atravesó los caminos previstos. Sin embargo, la evidencia de reparto *multipath* completo no se observó en todos los niveles de la jerarquía. Por tanto, su interpretación correcta es la de una exploración parcial y útil, no la de una solución ECMP completa finalizada.

9.3. Lecciones técnicas sobre RESTCONF y OVS

Uno de los aprendizajes más importantes del proyecto es que la validación de una solución SDN no puede basarse únicamente en la respuesta del controlador. En este trabajo, la instalación mediante RESTCONF se consideró válida solo cuando pudo contrastarse con evidencias del *datapath*: flujos presentes en Open vSwitch, contadores no nulos, conectividad extremo a extremo y resultados de tráfico coherentes.

La Tabla 9.3 resume las principales lecciones técnicas derivadas del uso conjunto de OpenDaylight, RESTCONF y Open vSwitch.

Tabla 9.3: Lecciones sobre OpenDaylight, RESTCONF y Open vSwitch

Lección técnica	**Evidencia observada**	**Implicación metodológica**
RESTCONF requiere verificación posterior	Una operación aceptada por la controladora no basta para demostrar funcionamiento en el *datapath*.	Toda instalación debe contrastarse con OVS, contadores y tráfico real.
El inventario de OpenDaylight no equivale a tráfico efectivo	Los nodos y grupos pueden aparecer en el inventario sin que todos los caminos se utilicen.	La validación debe distinguir entre descubrimiento, instalación y uso efectivo.
Los contadores por *bucket* son esenciales en ECMP	La presencia de grupos *select* no demuestra por sí sola reparto de tráfico.	Para hablar de *multipath* es necesario comprobar actividad real en los *buckets*.
La automatización en Python mejora la trazabilidad	Las campañas finales se apoyan en scripts, manifiestos, tablas y salidas reproducibles.	Reduce el riesgo de depender de observaciones manuales o estados no repetibles.

Esta distinción entre plano de control y plano de datos es especialmente relevante en las extensiones ECMP. La presencia de un grupo *select* en OpenDaylight o en OVS no implica por sí sola que exista reparto de tráfico. Para afirmar *multipath* es necesario comprobar que los *buckets* asociados reciben tráfico real. Por ello, los contadores de grupo y de *bucket* se convierten en una evidencia más fuerte que la mera instalación de la regla.

También se confirma la utilidad de mantener una automatización prioritaria en Python. Las primeras pruebas manuales fueron útiles para diagnosticar problemas, pero los resultados integrables en la memoria deben proceder de *scripts* reproducibles, con rutas estables, manifiestos y salidas verificables. Esta decisión reduce el riesgo de mezclar estados experimentales y facilita la revisión posterior del proyecto.

9.4. Interpretación del intento hacia ECMP completo

La campaña exploratoria final del Capítulo 8 respondió a una pregunta técnica razonable: si el demostrador ECMP limitado funcionaba, ¿era posible extender esa lógica a un intento más completo sobre la topología Fat-Tree k=4? La respuesta obtenida es matizada. El prototipo pudo instalar grupos *select* en *switches* de borde y agregación, pudo migrarse a RESTCONF y pudo transportar tráfico punto a punto y de estrés. No obstante, la activación de varios *buckets* se observó de forma clara en los *switches* de borde, pero no de forma completa en los *switches* de agregación.

La Tabla 9.4 recoge las principales limitaciones observadas y su interpretación metodológica.

Tabla 9.4: Limitaciones del intento hacia ECMP completo

Limitación	**Evidencia observada**	**Interpretación metodológica**
Cobertura limitada de pares	La exploración se centró en pares concretos, como `h1`–`h16` y `h2`–`h15`.	No permite afirmar ECMP completo para todos los pares de hosts.
Activación incompleta en agregación	Los switches de borde activaron varios *buckets*, pero los de agregación no mostraron reparto completo.	El resultado debe calificarse como parcial y no como ECMP completo.
Dependencia del patrón de tráfico	La selección de *buckets* puede depender del *hash*, del número de flujos y de las cabeceras utilizadas.	Una matriz de tráfico más amplia podría producir comportamientos distintos.
Complejidad de control	La combinación de reglas deterministas, grupos *select* y rutas jerárquicas aumenta la dificultad de verificación.	La implementación completa requiere un diseño de control más sistemático.

Este resultado no debe considerarse un fallo del proyecto. Al contrario, tiene valor porque delimita con precisión qué se ha conseguido y qué no. Se ha demostrado que la instalación RESTCONF de grupos y flujos es viable en un escenario jerárquico más complejo que el ECMP limitado. También se ha demostrado que el tráfico puede circular de forma estable y que existe *multipath* observable en borde. Sin embargo, no se ha demostrado que todos los niveles de la Fat-Tree participen en un reparto ECMP completo.

La causa puede estar relacionada con varios factores: el comportamiento de selección de los grupos *select*, la polarización del *hash* por flujo, la matriz de tráfico limitada, la forma en que se fijaron los caminos aguas abajo o la propia dificultad de controlar ECMP completo con OpenFlow 1.3 en un entorno emulado. En cualquier caso, la consecuencia metodológica es clara: el resultado debe formularse como intento exploratorio parcial y no como ECMP completo validado.

9.5. Relación entre Leaf-Spine y Fat-Tree

La inclusión de Leaf-Spine como etapa intermedia fue una decisión metodológica acertada. Antes de abordar Fat-Tree, permitió comprobar en un *fabric* más simple qué significa disponer de caminos alternativos y cómo cambia el comportamiento del tráfico cuando se pasa de un único camino a una estrategia *multipath*. Esta etapa ayudó a fijar la intuición y el criterio metodológico que posteriormente se aplicaron a Fat-Tree, pero no debe interpretarse como una campaña directamente comparable en términos de *throughput* absoluto ni como una demostración previa de ECMP completo en una arquitectura jerárquica.

La Tabla 9.5 resume el papel comparado de Leaf-Spine y Fat-Tree dentro del proyecto.

Tabla 9.5: Relación entre Leaf-Spine y Fat-Tree

Aspecto	**Leaf-Spine**	**Fat-Tree**	**Aprendizaje**
Complejidad estructural	Arquitectura más simple, con redundancia ascendente fácil de visualizar.	Arquitectura jerárquica con borde, agregación y *core*.	La lógica *multipath* se vuelve más difícil de validar al aumentar las capas.
Uso de redundancia	Permite observar de forma clara la diferencia entre camino único y ECMP.	Requiere coordinar varios niveles de decisión y verificación.	El éxito en Leaf-Spine no garantiza ECMP completo en Fat-Tree.
Papel metodológico	Etapa intermedia para validar intuiciones y mecanismos básicos.	Escenario principal para el diseño y evaluación final.	La progresión incremental mejora la defendibilidad del proyecto.
Interpretación	Los resultados ayudan a comprender el valor de explotar caminos equivalentes.	Los resultados muestran tanto la viabilidad como los límites de esa explotación.	La narrativa final debe distinguir entre demostración, extensión acotada e intento exploratorio.

La comparación entre Leaf-Spine y Fat-Tree realizada en este trabajo es, por tanto, metodológica y estructural. En Leaf-Spine se limitaron los enlaces ascendentes para hacer visible el efecto de utilizar uno o varios *uplinks* bajo saturación controlada. En Fat-Tree, la campaña principal evaluó escenarios representativos sobre una arquitectura jerárquica mediante *forwarding* determinista instalado desde OpenDaylight RESTCONF. Por este motivo, los valores de *throughput* de ambos bloques no deben leerse como una clasificación de rendimiento entre topologías, sino como evidencias obtenidas bajo objetivos experimentales distintos.

La diferencia principal entre ambas etapas es la escala de complejidad. En Leaf-Spine, la redundancia puede explicarse de forma relativamente directa mediante *uplinks* equivalentes. En Fat-Tree, en cambio, la redundancia se distribuye en varias capas y depende de decisiones coordinadas entre borde, agregación y *core*. Por ello, que ECMP funcione de forma controlada en Leaf-Spine no implica automáticamente que se obtenga ECMP completo en Fat-Tree.

Esta progresión refuerza la narrativa global del TFG. El proyecto no salta directamente de topologías simples a una arquitectura compleja, sino que construye una cadena de validación incremental: banco de pruebas, topologías básicas, Leaf-Spine, Fat-Tree determinista, ECMP limitado y exploración avanzada. Esa secuencia facilita justificar tanto los resultados positivos como los límites encontrados.

9.6. Amenazas a la validez

Las conclusiones de este trabajo deben interpretarse dentro del alcance definido por el banco de pruebas. La topología se ejecuta sobre Mininet y Open vSwitch en una única máquina física, por lo que las medidas de rendimiento no representan prestaciones absolutas extrapolables a hardware real. Su valor principal reside en la comparación interna entre escenarios ejecutados bajo condiciones equivalentes.

Una amenaza adicional es comparar de forma directa campañas que no fueron diseñadas con las mismas condiciones. En particular, los resultados Leaf-Spine y Fat-Tree no comparten el mismo objetivo experimental ni la misma configuración de capacidad de enlace. Por ello, la discusión evita extraer conclusiones del tipo "una topología rinde más que otra" a partir de valores absolutos de *throughput*; en su lugar, compara el papel metodológico de cada bloque, su complejidad estructural y el grado de evidencia conseguido.

La Tabla 9.6 recoge las principales amenazas a la validez y las mitigaciones aplicadas.

Tabla 9.6: Amenazas a la validez

Amenaza	**Impacto potencial**	**Mitigación aplicada**
Entorno emulado	Los valores absolutos de *throughput* y latencia dependen de la máquina anfitriona y de Mininet/OVS.	Los resultados se interpretan como comparación interna, no como prestaciones extrapolables a hardware real.
Escala limitada k=4	La instancia no representa una red de centro de datos a gran escala.	Se presenta como escala manejable para validación académica y no como despliegue productivo.
Sobreinterpretación de ECMP	La existencia de grupos *select* podría confundirse con balanceo completo.	Se revisan contadores por *bucket* y se distingue entre instalación, tráfico y reparto efectivo.
Dependencia del tráfico generado	El comportamiento puede variar con otros pares de hosts, número de flujos o protocolos.	Se delimitan claramente los escenarios evaluados y se evita generalizar más allá de las evidencias.
Estado residual del banco de pruebas	Reglas o configuraciones anteriores podrían contaminar una ejecución.	Se emplean procedimientos de limpieza, instalación controlada y validación posterior.

La amenaza más importante para la interpretación de los resultados ECMP es la sobreinterpretación. La existencia de grupos *select* no basta para afirmar balanceo completo, y la presencia de tráfico en un grupo no implica necesariamente que todos sus *buckets* hayan sido utilizados. Por este motivo, el análisis se apoya en contadores por *bucket* y distingue explícitamente entre instalación, propagación, tráfico efectivo y reparto observado.

Otra amenaza relevante es la escala. Fat-Tree k=4 es una instancia manejable y adecuada para un TFG, pero no representa por sí sola una infraestructura de centro de datos a gran escala. La implementación demuestra principios arquitectónicos y de control, pero no pretende caracterizar exhaustivamente el comportamiento de una red de producción.

9.7. Lecciones aprendidas

La Tabla 9.7 sintetiza las lecciones principales derivadas de la implementación y evaluación del banco de pruebas.

Tabla 9.7: Lecciones aprendidas

Lección	Evidencia en el proyecto	Consecuencia para la defensa del TFG
La validación incremental es necesaria	El proyecto avanzó desde topologías básicas hasta Leaf-Spine y Fat-Tree.	Permite justificar que cada bloque se apoya en evidencias previas.
La trazabilidad debe cubrir control y datos	RESTCONF, OVS, contadores, pings e `iperf3` se contrastan de forma conjunta.	Refuerza la credibilidad de los resultados frente a una validación superficial.
La simplicidad puede ser metodológicamente valiosa	La campaña determinista es menos ambiciosa que ECMP completo, pero más estable y verificable.	Permite defender una campaña principal sólida sin sobreprometer resultados.
Los resultados parciales también aportan valor	El intento hacia ECMP completo no cierra ECMP generalizado, pero identifica límites técnicos reales.	Ayuda a formular trabajo futuro de forma concreta y fundamentada.
La narrativa debe separar resultados y exploraciones	La campaña RESTCONF determinista, el ECMP limitado y el intento hacia ECMP completo tienen alcances diferentes.	Evita contradicciones y mantiene una interpretación académicamente honesta.

La primera lección es que la validación incremental ha sido imprescindible. Cada bloque del proyecto ha reducido la incertidumbre del siguiente: las topologías básicas validaron el entorno, Leaf-Spine validó la lógica *multipath* en un *fabric* más simple, y Fat-Tree permitió evaluar una arquitectura jerárquica más rica.

La segunda lección es que en SDN la trazabilidad debe verificarse en ambos planos. No basta con que OpenDaylight acepte una operación RESTCONF; es necesario comprobar que la regla aparece en el *datapath* y que los contadores confirman su uso. Esta idea aparece de forma transversal en todo el proyecto y es una de las bases de su defendibilidad.

La tercera lección es que un resultado parcial puede ser valioso si se formula correctamente. El intento exploratorio hacia ECMP completo no demuestra ECMP completo, pero sí documenta un intento técnicamente razonado, reproducible y delimitado. Esa evidencia permite justificar por qué ECMP completo queda como trabajo futuro y evita presentar una conclusión no respaldada por los datos.

9.8. Cierre de la discusión

La discusión permite extraer una conclusión central: el banco de pruebas SDN desarrollado es funcional, trazable y suficientemente flexible para evaluar una topología Fat-Tree k=4 con políticas de *forwarding* instaladas desde OpenDaylight. La campaña determinista constituye el resultado principal porque combina cobertura experimental, estabilidad y reproducibilidad. Las extensiones ECMP añaden valor como pruebas técnicas: el ECMP limitado queda validado en un escenario controlado, mientras que el intento hacia ECMP completo identifica una frontera técnica que no debe sobrepasarse en la interpretación.

Por tanto, el proyecto no debe presentarse como una implementación final de ECMP completo en Fat-Tree, sino como una implementación SDN reproducible de una arquitectura Fat-Tree multidimensional, con una campaña principal sólida y una exploración *multipath* honesta y acotada. Esta lectura conserva el rigor metodológico del trabajo y prepara de forma natural las conclusiones y líneas futuras del capítulo siguiente.

10. CONCLUSIONES Y TRABAJO FUTURO

Este capítulo cierra la memoria sintetizando el grado de cumplimiento del trabajo, la relación entre los resultados obtenidos y la propuesta inicial, las conclusiones técnicas derivadas de la implementación y las principales líneas de trabajo futuro. El objetivo no es introducir nuevas evidencias experimentales, sino ordenar de forma crítica las evidencias ya presentadas en los capítulos anteriores y formular una lectura final defendible del proyecto.

El resultado principal del TFG es la construcción y validación de un banco de pruebas SDN reproducible para una topología Fat-Tree k=4, basado en Mininet, Open vSwitch, OpenFlow 1.3 y OpenDaylight. La campaña experimental principal se ha apoyado en *forwarding* determinista instalado mediante RESTCONF, con verificación posterior en el plano de datos. Sobre esa base se han añadido extensiones *multipath*: primero un demostrador ECMP limitado y posteriormente un intento exploratorio hacia ECMP completo. Esta jerarquía de evidencias permite cerrar el trabajo sin sobreinterpretar los resultados obtenidos.

10.1. Balance general del trabajo

El desarrollo del TFG ha seguido una estrategia incremental. En primer lugar, se construyó el banco de pruebas SDN y se validó su funcionamiento mediante topologías de complejidad reducida. Posteriormente, se utilizó Leaf-Spine como etapa metodológica intermedia para introducir la lógica *multipath* en un entorno controlado. Finalmente, se diseñó e implementó una topología Fat-Tree k=4, que constituye la arquitectura principal de evaluación experimental.

La decisión de no abordar directamente Fat-Tree desde el inicio ha sido metodológicamente acertada. La validación progresiva permitió reducir la incertidumbre técnica, identificar problemas de conectividad, controlar la instalación de reglas y consolidar una forma de trabajo basada en evidencias: inventarios, reglas instaladas, contadores del *datapath*, pruebas de conectividad, resultados de tráfico, tablas y figuras generadas de forma reproducible.

El trabajo también ha requerido acotar el alcance frente a la propuesta inicial. Algunas líneas previstas o sugeridas, como Containernet, servicios Docker, modificaciones topológicas tipo *subways*/*superways* o reconfiguración desatendida, no se han incorporado a la campaña experimental final. Esta decisión no se interpreta como una pérdida de rigor, sino como una delimitación necesaria para priorizar una contribución principal estable, trazable y defendible.

10.2. Cumplimiento de objetivos

La Tabla 10.1 resume el grado de cumplimiento de los objetivos específicos planteados en el Capítulo 1. Esta tabla cierra la trazabilidad iniciada al comienzo de la memoria y permite distinguir entre objetivos plenamente cumplidos, objetivos cumplidos parcialmente y resultados exploratorios.

Tabla 10.1: Cumplimiento final de objetivos

Obj.	Objetivo	Estado	Evidencia principal	Conclusión defendible
OE1	Construir un banco de pruebas SDN basado en Linux, Mininet, Open vSwitch, OpenFlow y OpenDaylight.	Cumplido	Capítulos 3 y 4: arquitectura del entorno, conexión con OpenDaylight, configuración OVS/OpenFlow y validación incremental.	El entorno queda operativo, documentado y reproducible para las campañas posteriores.
OE2	Validar progresivamente el entorno mediante topologías de complejidad creciente antes de abordar Fat-Tree.	Cumplido	Capítulos 4 y 5: topologías básica, intermedia y Leaf-Spine como etapa metodológica.	La progresión evita saltar directamente a Fat-Tree sin evidencias previas del banco de pruebas.
OE3	Diseñar e implementar una topología Fat-Tree parametrizable, empleando k=4 como instancia experimental manejable.	Cumplido	Capítulo 6: generador Fat-Tree, nomenclatura, direccionamiento, puertos y análisis estructural.	La instancia k=4 permite evaluar una arquitectura jerárquica completa dentro del entorno local de emulación.
OE4	Instalar políticas de *forwarding* desde la controladora SDN y verificar su efecto real en el plano de datos.	Cumplido	Capítulos 3, 7 y 8: instalación mediante OpenDaylight RESTCONF y verificación posterior en Open vSwitch.	La campaña principal valida el flujo controlador–*datapath* con evidencias de reglas, contadores y conectividad.
OE5	Evaluar la topología Fat-Tree con métricas de latencia, rendimiento y evidencias de tráfico reproducibles.	Cumplido	Capítulos 7 y 8: ensayos punto a punto, tráfico concurrente distribuido, *hotspot*, RTT, *throughput* y contadores.	Las métricas permiten interpretar el funcionamiento interno de la campaña Fat-Tree RESTCONF bajo condiciones de emulación.
OE6	Explorar mecanismos *multipath* mediante grupos OpenFlow *select*, delimitando con precisión su alcance experimental.	Cumplido parcial	Capítulos 8 y 9: demostrador ECMP limitado, instalación RESTCONF de grupos *select* e intento exploratorio hacia ECMP completo.	El ECMP limitado queda validado en escenarios acotados; el ECMP completo en toda la jerarquía Fat-Tree queda como resultado parcial y trabajo futuro.
OE7	Documentar las limitaciones del banco de pruebas y formular líneas futuras coherentes con la propuesta inicial.	Cumplido	Capítulos 7, 8, 9 y 10: limitaciones de emulación, alcance de RESTCONF, discusión crítica y líneas futuras.	El trabajo distingue resultados principales, extensiones acotadas y líneas no implementadas sin sobreinterpretar las evidencias.

El objetivo relativo a la construcción del banco de pruebas SDN se considera cumplido. El entorno final combina Mininet, Open vSwitch, OpenFlow 1.3 y OpenDaylight, y se apoya en una estructura de proyecto reproducible. La validación incremental permitió confirmar que el banco de pruebas era funcional antes de abordar topologías de mayor complejidad.

También se considera cumplido el objetivo de diseñar e implementar una topología Fat-Tree parametrizable. La instancia k=4 utilizada en la campaña experimental proporciona una arquitectura completa, con *hosts*, *switches* de borde, *switches* de agregación, *switches core* y comunicación entre *pods*. Aunque la evaluación experimental se centra en k=4, el diseño y el análisis estructural permiten razonar sobre la escalabilidad de la topología.

El objetivo relativo a la instalación de políticas de *forwarding* desde la controladora SDN se cumple mediante la campaña RESTCONF determinista. Esta campaña constituye la evidencia principal del trabajo, porque valida no solo la existencia de la topología, sino también la capacidad de instalar reglas desde OpenDaylight, comprobar su presencia en Open vSwitch y ejecutar escenarios de tráfico representativos.

El objetivo asociado a mecanismos *multipath* debe formularse de forma más matizada. El demostrador ECMP limitado queda validado en escenarios concretos, incluyendo una variante instalada mediante RESTCONF. Sin embargo, el intento hacia ECMP completo en toda la jerarquía Fat-Tree no alcanza evidencia suficiente para considerarse un resultado cerrado. Por tanto, este objetivo se considera cumplido parcialmente: se ha explorado con rigor y se han documentado sus límites, pero queda como línea futura para una implementación completa.

10.3. Cumplimiento frente a la propuesta inicial

La propuesta inicial del TFG planteaba un marco amplio de trabajo alrededor de SDN, topologías de centro de datos, automatización, Fat-Tree, Leaf-Spine, métricas de red y posibles extensiones relacionadas con variantes topológicas o reconfiguración. Durante el desarrollo del proyecto, ese marco se concretó en una línea principal: construir una arquitectura SDN reproducible sobre Fat-Tree k=4, validarla experimentalmente y explorar de forma acotada mecanismos *multipath*.

La Tabla 10.2 recoge la trazabilidad final entre las premisas de la propuesta original y el tratamiento adoptado en la memoria. Su finalidad es hacer explícito qué elementos se han implementado, cuáles se han abordado parcialmente y cuáles quedan como trabajo futuro.

Tabla 10.2: Trazabilidad frente a la propuesta original

Premisa de la propuesta	**Tratamiento final**	**Estado**	**Justificación**
SDN con Mininet, Open vSwitch y OpenDaylight	Implementado como banco de pruebas principal.	Implementado	La arquitectura final combina Mininet, OVS, OpenFlow 1.3 y OpenDaylight con validación incremental.
Automatización en Python	Implementada para topologías, instalación, validación, procesado y generación de artefactos.	Implementado	La automatización permite reproducibilidad y reduce la dependencia de operaciones manuales extensas.
Topología Leaf-Spine	Utilizada como etapa metodológica intermedia.	Implementado	Permitió validar lógica *multipath* en un entorno reducido con saturación controlada.
Topología Fat-Tree	Diseñada, implementada y evaluada como arquitectura principal.	Implementado	Fat-Tree k=4 constituye el núcleo experimental de los capítulos 6–8.
Parametrización por k	Implementada en el generador y complementada con análisis teórico de escalabilidad.	Implementado parcialmente	Se evalúa experimentalmente k=4; valores superiores se analizan estructuralmente y quedan como extensión experimental futura.

Continúa en la página siguiente

Tabla 10.2 – Continuación

Premisa de la propuesta	**Tratamiento final**	**Estado**	**Justificación**
Métricas de latencia y *throughput*	Utilizadas en campañas Leaf-Spine y Fat-Tree.	Implementado	Las métricas se interpretan dentro de cada campaña y no como comparación absoluta entre topologías distintas.
Contadores y evidencias del *datapath*	Incorporados como criterio de validación de flujos, grupos y *buckets*.	Implementado	Los *dumps* y contadores de OVS son esenciales para verificar que las reglas instaladas transportan tráfico real.
RESTCONF y controladora SDN	Adoptados como eje de la campaña principal Fat-Tree.	Implementado	OpenDaylight RESTCONF instala el *forwarding* determinista principal y parte de las extensiones ECMP limitadas.
ECMP limitado mediante grupos *select*	Implementado como demostrador acotado sobre Fat-Tree.	Implementado	Se validan ramas equivalentes, grupos *select*, tráfico funcional y evidencias de contadores en escenarios concretos.
ECMP completo Fat-Tree	Abordado como intento exploratorio controlado.	Explorado	La migración RESTCONF fue viable, pero no se obtuvo evidencia suficiente de reparto completo en agregación/*core*.
Containernet y Docker	No integrados en la campaña experimental final.	Trabajo futuro	Añaden una capa de servicios y dependencias que excede el cierre experimental defendible del TFG.
Variantes topológicas *subways*/*superways*	No implementadas como modificación topológica real.	Trabajo futuro	Requerirían rediseño topológico, nuevas rutas y repetición de campañas consolidadas.
Reconfiguración desatendida	No implementada como sistema automático completo.	Trabajo futuro	Exige telemetría, lógica de decisión y actuación dinámica que se apoyaría en las evidencias ya generadas.
Escalabilidad más allá de k=4	Analizada teóricamente, no evaluada experimentalmente.	Trabajo futuro	La evaluación de $k > 4$ requiere más recursos, más reglas y una campaña específica de escalabilidad.

La mayor parte de los elementos nucleares de la propuesta se han implementado: SDN, Mininet, Open vSwitch, OpenDaylight, automatización en Python, Leaf-Spine, Fat-Tree, métricas de red, contadores del *datapath* y RESTCONF. Estos elementos forman la base real del banco de pruebas y de las campañas experimentales.

Otros elementos se han tratado de forma parcial o exploratoria. La parametrización por k existe en el diseño y se complementa con análisis teórico, pero la evaluación experimental se limita a k=4. El ECMP completo en Fat-Tree se ha intentado de forma controlada, pero no se presenta como resultado final porque la activación *multipath* completa en agregación y *core* no quedó demostrada.

Finalmente, Containernet, Docker, *subways*, *superways* y reconfiguración desatendida quedan como líneas futuras. Incorporarlas dentro de la campaña final habría exigido nuevas fases de diseño, nuevas dependencias, repetición de experimentos y nuevos criterios de validación. Su exclusión se justifica por la necesidad de cerrar una contribución principal sólida antes de ampliar el alcance.

10.4. Conclusiones técnicas

La Tabla 10.3 sintetiza el papel de cada bloque experimental y metodológico desarrollado a lo largo de la memoria. Esta visión de conjunto permite entender que el TFG no se apoya en una única prueba aislada, sino en una cadena de validación progresiva.

Tabla 10.3: Síntesis final por bloques

Bloque	**Resultado principal**	**Evidencia generada**	**Alcance real**
Banco de pruebas y arquitectura	Entorno SDN reproducible con Mininet, OVS, OpenFlow 1.3 y OpenDaylight.	Versiones, estructura del proyecto, *scripts* de automatización, validaciones y listados RESTCONF/OVS.	Base técnica del TFG; no constituye por sí sola una evaluación de rendimiento.
Validación básica e intermedia	Comprobación incremental de conectividad, reglas y contadores.	Topologías simples, validación de *flows*, *dumps* y microvalidación cuantitativa.	Reduce el riesgo antes de pasar a arquitecturas de centro de datos.
Leaf-Spine	Validación metodológica de *multipath* en topología reducida.	Comparaciones *single_path*/ECMP y L3 estático/ECMP bajo enlaces limitados a 100 Mbps.	Etapa intermedia; no comparable de forma absoluta con Fat-Tree.
Diseño Fat-Tree	Implementación parametrizada y validación estructural de Fat-Tree k=4.	Generador Python, metadatos, direccionamiento, puertos, validación SDN y escalabilidad teórica.	Arquitectura principal del trabajo experimental.
Metodología experimental	Definición de escenarios, métricas, procedimiento y limitaciones.	Matriz experimental, métricas, tratamiento de resultados y amenazas a la validez.	Marco de interpretación para evitar comparaciones no equivalentes.
RESTCONF determinista	Campaña principal Fat-Tree con *forwarding* instalado desde OpenDaylight RESTCONF.	Instalación de reglas, *preflight*, ensayos punto a punto, concurrente y *hotspot*, tablas y figuras.	Resultado principal, completo y trazable del Capítulo 8.
ECMP limitado	Demostrador acotado de grupos OpenFlow *select* en Fat-Tree.	Instalación en *datapath*, migración RESTCONF, uso de ramas equivalentes y contadores de *buckets*.	Extensión validada, pero no generalizable a ECMP completo en toda la jerarquía.
Intento hacia ECMP completo	Exploración controlada de un mecanismo más ambicioso en Fat-Tree k=4.	Tráfico válido, migración RESTCONF viable y *multipath* en borde; activación insuficiente en agregación/*core*.	Resultado parcial documentado honestamente como frontera técnica y trabajo futuro.
Discusión crítica	Interpretación conjunta de las evidencias y sus límites.	Síntesis de estrategias, amenazas a la validez, relación Leaf-Spine/Fat-Tree y lecciones aprendidas.	Cierre metodológico previo a conclusiones.

La primera conclusión técnica es que el banco de pruebas SDN desarrollado es funcional y reproducible. La combinación de Mininet, OVS, OpenFlow y OpenDaylight permite instanciar topologías, programar reglas de *forwarding*, consultar el estado del sistema y obtener evidencias verificables del plano de datos. Esta trazabilidad ha sido esencial para evitar que la validación dependa únicamente de que una operación sea aceptada por la controladora.

La segunda conclusión es que Fat-Tree k=4 constituye una instancia experimental adecuada para un TFG. Su tamaño permite estudiar una arquitectura jerárquica con *pods*, *switches* de borde, agregación y *core*, pero sin superar los límites razonables de un entorno local de emulación. Esta elección ha permitido ejecutar campañas de tráfico, procesar resultados y presentar evidencias interpretables.

La tercera conclusión es que la campaña RESTCONF determinista debe considerarse el resultado principal del trabajo. Su valor no reside en demostrar balanceo automático, sino en validar una arquitectura Fat-Tree controlada desde una capa SDN, con reglas instaladas desde OpenDaylight y verificadas posteriormente en OVS. Esta campaña aporta la mayor cobertura experimental y sostiene la contribución principal de la memoria.

La cuarta conclusión es que los mecanismos *multipath* requieren una interpretación prudente. El demostrador ECMP limitado confirma que los grupos OpenFlow *select* pueden utilizarse de forma acotada en Fat-Tree y que es posible observar ramas equivalentes mediante contadores. Sin embargo, el intento hacia ECMP completo muestra que extender esta lógica a todos los niveles de la jerarquía es más complejo. La existencia de caminos alternativos o de grupos instalados no basta para afirmar balanceo completo.

La quinta conclusión es que los resultados cuantitativos deben interpretarse dentro de cada campaña. Los valores obtenidos en Leaf-Spine y Fat-Tree no son comparables como medidas absolutas de capacidad, porque responden a configuraciones, objetivos y condiciones experimentales diferentes. Su valor reside en la comparación interna, en las tendencias observadas y en la relación entre las evidencias generadas y el alcance declarado.

10.5. Limitaciones finales

La principal limitación del trabajo deriva del uso de un entorno emulado. Mininet y Open vSwitch se ejecutan sobre una misma máquina física, por lo que las medidas de rendimiento pueden verse afectadas por CPU, memoria, planificación del sistema operativo y carga local. Por esta razón, los valores de *throughput* y RTT no deben interpretarse como prestaciones absolutas extrapolables a una infraestructura física de centro de datos.

Una segunda limitación es el tamaño de la instancia evaluada. Fat-Tree k=4 permite estudiar una arquitectura jerárquica completa y manejable, pero no caracteriza por sí sola el comportamiento de valores superiores de k. El análisis teórico de escalabilidad permite razonar sobre el crecimiento estructural, aunque no sustituye a una campaña experimental específica con $k > 4$.

Una tercera limitación está relacionada con la naturaleza del *forwarding*. La campaña principal se basa en caminos deterministas instalados mediante RESTCONF. Esto aporta trazabilidad, pero no implementa por sí mismo un sistema autónomo de optimización SDN. Las extensiones ECMP demuestran posibilidades acotadas, pero no constituyen una solución completa de balanceo en toda la jerarquía Fat-Tree.

Finalmente, el intento hacia ECMP completo debe interpretarse como resultado parcial. Se logró instalar grupos y flujos, migrar la lógica a RESTCONF y transportar tráfico de forma estable, pero no se obtuvo evidencia suficiente de activación equilibrada de *buckets*

en todos los niveles de la topología. Esta limitación no invalida el trabajo; al contrario, delimita con honestidad una frontera técnica que puede ser abordada en investigaciones posteriores.

10.6. Trabajo futuro

La Tabla 10.4 prioriza las líneas de trabajo futuro derivadas del proyecto. La priorización se basa en dos criterios: la relación directa con las limitaciones observadas y la coherencia con la propuesta inicial del TFG.

Tabla 10.4: Priorización de trabajo futuro

Línea futura	**Prioridad**	**Dependencia técnica**	**Motivación**
ECMP completo Fat-Tree con política de selección más robusta	Alta	Modelo de rutas, grupos *select*, verificación de *buckets* en borde, agregación y *core*.	Permitiría convertir el intento exploratorio parcial en una validación *multipath* más completa.
Telemetría y reconfiguración desatendida	Alta	Contadores fiables, detección de congestión, decisión de control y reinstalación de reglas.	Conecta directamente con la propuesta inicial y con las evidencias de contadores obtenidas.
Evaluación experimental con $k > 4$	Alta	Optimización de recursos, generación de reglas a mayor escala y campañas de validación específicas.	Completaría el análisis de escalabilidad más allá de la instancia k=4.
Containernet y servicios contenerizados	Media	Integración de contenedores, servicios reales y posible aislamiento de cargas.	Aportaría escenarios de aplicación más cercanos a servicios desplegados sobre el centro de datos emulado.
Variantes topológicas *subways*/*superways*	Media	Rediseño de enlaces, generación de rutas, validación comparativa y repetición de campañas.	Permitiría estudiar mejoras estructurales o caminos añadidos sobre la Fat-Tree base.
Comparativa con hardware físico o entorno distribuido	Media	Disponibilidad de infraestructura, *switches* compatibles o varios nodos físicos.	Ayudaría a separar limitaciones propias de Mininet/OVS de comportamientos de red física.
Tratamiento estadístico más amplio	Media	Más repeticiones, más cargas, semillas controladas y análisis de intervalos de confianza.	Refinaría la solidez estadística de las conclusiones cuantitativas.
Integración de visualización operativa en tiempo real	Baja	Telemetría periódica, paneles de visualización y almacenamiento temporal de métricas.	Mejoraría la observabilidad del banco de pruebas, aunque no es imprescindible para validar la arquitectura base.

La primera línea prioritaria es profundizar en ECMP completo sobre Fat-Tree. Para ello sería necesario diseñar una política de selección más robusta, ampliar la matriz de tráfico, estudiar el comportamiento del *hash* por flujo y validar de forma explícita la activación de *buckets* en borde, agregación y *core*. Esta línea permitiría transformar el intento exploratorio parcial en una validación *multipath* más completa.

La segunda línea prioritaria es incorporar telemetría y reconfiguración desatendida. El trabajo actual genera contadores y evidencias de tráfico, pero no implementa un ciclo automático de detección, decisión y actuación. Una extensión natural sería usar esas métricas para detectar congestión o infrautilización y reinstalar reglas mediante RESTCONF.

La tercera línea es ampliar la evaluación experimental a valores de $k > 4$. Esto permitiría comprobar cómo crecen el número de reglas, grupos, contadores y artefactos generados, así como identificar los límites prácticos del entorno emulado. Esta ampliación debería abordarse con una metodología específica, ya que el aumento de escala incrementa significativamente la complejidad operativa.

Otras líneas futuras incluyen la integración de Containernet y servicios contenerizados, la evaluación de variantes topológicas tipo *subways*/*superways*, la comparación con hardware físico o entornos distribuidos y la ampliación del tratamiento estadístico. Todas ellas son coherentes con la propuesta inicial, pero requieren una fase adicional de diseño y validación para no comprometer la defendibilidad del trabajo ya consolidado.

10.7. Cierre final

El TFG ha cumplido su objetivo principal: diseñar, implementar y evaluar un banco de pruebas SDN reproducible para una arquitectura Fat-Tree k=4. La memoria documenta no solo el resultado final, sino también la progresión metodológica que permitió alcanzarlo: validación incremental, etapa Leaf-Spine, diseño Fat-Tree, metodología experimental, campaña RESTCONF determinista, demostrador ECMP limitado e intento exploratorio hacia ECMP completo.

La contribución más sólida del trabajo es la campaña Fat-Tree basada en OpenDaylight RESTCONF y verificada en Open vSwitch. Esta campaña demuestra que es posible construir una arquitectura jerárquica controlada mediante SDN, instalar reglas trazables y obtener evidencias reproducibles de conectividad, tráfico y contadores. Sobre esa base, las extensiones *multipath* aportan valor adicional, siempre que se interpreten dentro de su alcance real.

El proyecto no debe presentarse como una implementación final de ECMP completo ni como una evaluación de prestaciones físicas de una red de producción. Su aportación es más precisa y defendible: una implementación SDN reproducible de Fat-Tree, una metodología experimental trazable y una exploración honesta de mecanismos *multipath* que identifica tanto posibilidades como límites. Esta lectura permite cerrar el trabajo con rigor y abre líneas futuras claras para continuar la investigación.

A. ENTORNO DE TRABAJO

Este apéndice recoge información operativa complementaria sobre el entorno de trabajo utilizado durante el desarrollo del TFG. Su objetivo no es repetir la arquitectura descrita en el Capítulo 3, sino dejar constancia de las rutas, variables y comandos mínimos que permiten reproducir la preparación básica del banco de pruebas.

El entorno experimental se construyó alrededor de una instalación local basada en Mininet, Open vSwitch, OpenFlow 1.3 y OpenDaylight. Sobre esta base se desarrollaron *scripts* Python para desplegar topologías, instalar reglas mediante RESTCONF, ejecutar campañas de tráfico, capturar evidencias y generar artefactos procesados. La Tabla A.1 resume las variables y rutas principales utilizadas durante las sesiones de trabajo.

Tabla A.1: Variables y rutas del entorno

Elemento	Función	Valor o ruta	Uso en el proyecto
`TFG_ROOT`	Directorio raíz del proyecto local.	`$HOME/Escritorio/TFG/TFG-SDN`	Permite ejecutar *scripts* y generar salidas con rutas reproducibles.
`PYTHONPATH`	Ruta de búsqueda de módulos Python.	`$TFG_ROOT:$PYTHONPATH`	Facilita la ejecución de módulos del repositorio desde la raíz del proyecto.
`ODL_HOST`	Dirección de la controladora OpenDaylight.	`127.0.0.1`	Identifica el punto de acceso RESTCONF y OpenFlow del banco de pruebas local.
`ODL_REST_PORT`	Puerto RESTCONF de OpenDaylight.	`8181`	Utilizado por los *scripts* que instalan o consultan información mediante RESTCONF.
`ODL_USER`	Usuario RESTCONF de OpenDaylight.	`admin`	Credencial usada por las operaciones RESTCONF del entorno de laboratorio.
`runs/`	Directorio de campañas experimentales.	`runs/ch08_fattree/`	Conserva salidas brutas, comandos Mininet, *dumps* OVS y resúmenes por campaña.
`data/processed/`	Directorio de datos procesados.	`data/processed/ch08_fattree/`	Almacena manifiestos, CSV consolidados y resultados derivados.
`figures/`	Directorio de figuras generadas o editoriales.	`figures/ch08_resultados/`	Recoge gráficos exportados en formatos integrables en Overleaf.
`tables/`	Directorio de tablas LaTeX.	`tables/chXX/`	Permite insertar tablas mediante `\input` y mantener trazabilidad por capítulo.
`listings/`	Directorio de listados LaTeX.	`listings/chXX/`	Contiene extractos representativos de comandos, salidas y secuencias reproducibles.

A.1. Preparación operativa

Antes de ejecutar las campañas experimentales, las sesiones de trabajo se inicializaron desde la raíz del proyecto y con variables de entorno homogéneas. Esta práctica redujo errores de ruta, facilitó la generación automática de salidas y permitió conservar los artefactos de cada campaña en directorios diferenciados.

La Tabla A.2 resume los comandos mínimos de preparación y comprobación del entorno. No representa una guía exhaustiva de instalación del sistema, sino una referencia operativa de los pasos más utilizados durante las sesiones experimentales.

Tabla A.2: Comandos operativos de reproducibilidad

Paso	Comando	Finalidad
Preparación del directorio	`cd $HOME/Escritorio/TFG/TFG-SDN`	Sitúa la sesión en la raíz del proyecto antes de ejecutar *scripts*.
Definir raíz del proyecto	`export TFG_ROOT="$HOME/Escritorio/ TFG/TFG-SDN"`	Homogeneiza rutas de entrada y salida.
Definir ruta Python	`export PYTHONPATH="$TFG_ROOT: $PYTHONPATH"`	Permite importar módulos del proyecto desde los *scripts* experimentales.
Credencial temporal de superusuario	`sudo -v`	Evita interrupciones interactivas durante comandos OVS y Mininet.
Arranque de OpenDaylight	`$HOME/odl/odl-titanium/ karaf-0.22.0/bin/karaf`	Activa la controladora SDN usada por RESTCONF y OpenFlow.
Arranque de Fat-Tree k=4	`sudo -E mn --custom tfg_sdn/mininet/topos/fattree.py --topo fattree,k=4 --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13 --mac`	Instancia la topología principal utilizada en la campaña Fat-Tree.
Comprobación de sintaxis de *scripts*	`python3 -m py_compile tfg_sdn/experiments/script.py`	Verifica que el *script* no contiene errores de sintaxis antes de ejecutarlo.
Localización de tablas generadas	`find tables -type f -name "*.tex"| sort`	Comprueba la existencia de artefactos LaTeX generados por los *scripts*.

A.2. Criterios mínimos de comprobación

En las sesiones de validación se siguieron varios criterios mínimos antes de considerar una campaña como utilizable. En primer lugar, la controladora OpenDaylight debía estar activa y accesible por RESTCONF. En segundo lugar, los *switches* Open vSwitch debían aparecer conectados a la controladora mediante OpenFlow 1.3. En tercer lugar, las reglas instaladas debían ser visibles en el *datapath* mediante volcados de OVS. Por último, la conectividad y el tráfico se consideraban válidos únicamente cuando estaban respaldados por pings, salidas de `iperf3`, contadores no nulos o manifiestos de procesado.

Estos criterios son coherentes con la metodología experimental descrita en el Capítulo 7. La finalidad principal fue evitar que la validación dependiera solo de que una orden fuera aceptada por la controladora, exigiendo también evidencias observables en el plano de datos.

B. SCRIPTS RELEVANTES

Este apéndice recoge una selección de *scripts* representativos del proyecto. No se incluye el código completo del repositorio, ya que ello aumentaría innecesariamente la extensión de la memoria y dificultaría su lectura. En su lugar, se presenta un inventario funcional que relaciona cada *script* con su papel dentro de la metodología experimental y con las evidencias generadas.

El criterio de selección ha sido incluir únicamente *scripts* que intervienen de forma directa en el despliegue de topologías, instalación de reglas, ejecución de campañas, procesado de resultados o generación de artefactos de discusión y conclusiones. Los *scripts* auxiliares, pruebas intermedias o versiones descartadas no se listan en este apéndice.

B.1. Inventario de scripts principales

La Tabla B.1 resume los *scripts* principales utilizados durante el desarrollo del TFG. Para cada uno se indica su ruta relativa dentro del proyecto, su función, el capítulo con el que se relaciona y la evidencia principal que contribuye a generar.

Tabla B.1: Inventario de *scripts* relevantes del proyecto

Ruta	Función	Capítulo	Evidencia asociada
`tfg_sdn/mininet/topos/fattree.py`	Genera la topología Fat-Tree parametrizable.	Capítulo 6	Instancia k=4, nomenclatura de *hosts* y *switches*, estructura por *pods*.
`tfg_sdn/experiments/ch08_install_restconf_forwarding.py`	Instala el *forwarding* determinista mediante OpenDaylight RESTCONF.	Capítulo 8	Reglas instaladas, verificación en OVS y campaña RESTCONF principal.
`tfg_sdn/experiments/ch08_restconf_preflight_validate.py`	Prepara y resume la validación previa de conectividad y contadores.	Capítulo 8	*Preflight* RESTCONF, pings de control y contadores no nulos.
`tfg_sdn/experiments/ch08_restconf_p2p_tests.py`	Ejecuta ensayos punto a punto sobre Fat-Tree.	Capítulo 8	Escenarios local, intra-*pod* e inter-*pod* con RTT y *throughput*.
`tfg_sdn/experiments/ch08_restconf_concurrent_tests.py`	Ejecuta tráfico concurrente distribuido.	Capítulo 8	Flujos simultáneos distribuidos y evidencia de tráfico agregado.
`tfg_sdn/experiments/ch08_restconf_hotspot_tests.py`	Ejecuta el escenario *hotspot* hacia un único destino.	Capítulo 8	Tráfico concentrado, pings previos y resultados de `iperf3`.
`tfg_sdn/experiments/ch08_process_restconf_results.py`	Procesa la campaña RESTCONF determinista.	Capítulo 8	CSV procesados, tablas, figuras y manifiestos de resultados.
`tfg_sdn/experiments/ch08_ecmp_limited_demo.py`	Implementa el demostrador ECMP limitado a nivel de *datapath*.	Capítulo 8	Comparación *single-path*/ECMP limitado y activación de ramas.
`tfg_sdn/experiments/ch08_restconf_group_select_demo.py`	Instala grupos OpenFlow *select* mediante RESTCONF en el demostrador limitado.	Capítulo 8	Grupos *select*, flujos asociados, tráfico y contadores de *buckets*.

Continúa en la página siguiente

Tabla B.1 – Continuación

Ruta	Función	Capítulo	Evidencia asociada
`tfg_sdn/experiments/ch08_ecmp_full_restconf_attempt.py`	Ejecuta el intento exploratorio hacia ECMP completo con RESTCONF.	Capítulo 8	Instalación de grupos y flujos, tráfico de control y evidencia parcial de *multipath*.
`tfg_sdn/experiments/ch08_process_ecmp_full_phase_d.py`	Procesa las evidencias del intento exploratorio hacia ECMP completo.	Capítulo 8	Tablas, figuras y síntesis del intento hacia ECMP completo.
`tfg_sdn/experiments/ch09_prepare_discussion_artifacts.py`	Genera tablas de discusión crítica.	Capítulo 9	Síntesis global, comparación de estrategias, amenazas a la validez y lecciones aprendidas.
`tfg_sdn/experiments/ch10_prepare_conclusions_artifacts.py`	Genera tablas de cierre y trazabilidad final.	Capítulo 10	Cumplimiento de objetivos, propuesta original, síntesis final y trabajo futuro.
`tfg_sdn/experiments/appendix_prepare_artifacts.py`	Genera tablas de apoyo para los apéndices.	Apéndices	Variables, comandos, *scripts* relevantes, campañas y rutas de artefactos.

B.2. Relación con la reproducibilidad

La organización de los *scripts* permite reconstruir la cadena experimental seguida en la memoria. En primer lugar, los *scripts* de topología definen la estructura de red que será instanciada en Mininet. A continuación, los *scripts* de instalación de *forwarding* programan reglas o grupos mediante OpenDaylight RESTCONF o mediante interacción directa con Open vSwitch en los demostradores acotados. Después, los *scripts* de campaña preparan comandos de Mininet, ejecutan pruebas de conectividad y generan resultados de tráfico. Finalmente, los *scripts* de procesado consolidan las salidas en tablas, figuras, listados y manifiestos.

Esta separación resulta importante porque evita mezclar en un único bloque operaciones de naturaleza distinta. La reproducibilidad no se apoya únicamente en conservar los datos finales, sino en mantener identificables las fases de generación, ejecución, captura y procesado. Por ello, los capítulos principales de la memoria solo incluyen extractos representativos, mientras que este apéndice conserva una visión global de los *scripts* más relevantes.

C. RESULTADOS ADICIONALES

Este apéndice documenta el catálogo de campañas y las rutas principales donde se conservan los artefactos generados durante el trabajo experimental. Su finalidad es reforzar la trazabilidad de los resultados presentados en los capítulos 8, 9 y 10, sin repetir las figuras, tablas o listados ya integrados en el cuerpo principal de la memoria.

Los resultados experimentales del TFG no se reducen a las gráficas finales. Cada campaña conserva salidas brutas, comandos ejecutados, ficheros JSON de `iperf3`, volcados de Open vSwitch, resúmenes de validación, datos procesados, tablas LaTeX, figuras y manifiestos. Esta estructura permite relacionar las conclusiones del documento con artefactos concretos generados durante el proceso experimental.

C.1. Catálogo de campañas experimentales

La Tabla C.1 resume las principales campañas y bloques experimentales conservados. Se incluyen la campaña principal RESTCONF determinista, las extensiones ECMP limitadas y el intento exploratorio hacia ECMP completo.

Tabla C.1: Catálogo de campañas y bloques experimentales conservados

Campaña o bloque	Tipo	Estado	Evidencia principal	Ruta base
`ch08_fattree_restconf_20260519_0957`	Campaña principal RESTCONF determinista	Validada	Instalación de reglas, *preflight*, punto a punto, concurrente y *hotspot*.	`runs/ch08_fattree/`
Preflight RESTCONF de la campaña principal	Validación previa	Validada	Conectividad planificada y contadores de reglas con tráfico.	`runs/ch08_fattree/.../ft8_2r_preflight/`
Ensayos punto a punto RESTCONF	Punto a punto	Validada	Escenarios local, intra-*pod* e inter-*pod* con pings e `iperf3`.	`runs/ch08_fattree/.../ft8_3r_p2p/`
Tráfico concurrente RESTCONF	Tráfico distribuido	Validada	Cuatro flujos simultáneos distribuidos sobre Fat-Tree.	`runs/ch08_fattree/.../ft8_4r_concurrent/`
Escenario *hotspot* RESTCONF	Tráfico concentrado	Validada	Cuatro fuentes hacia un mismo destino y contadores asociados.	`runs/ch08_fattree/.../ft8_5r_hotspot/`
`ch08_fattree_ecmp_limited_20260520_0812`	ECMP limitado en *datapath*	Validada	Comparación *single-path*/ECMP limitado y uso de dos ramas.	`runs/ch08_fattree/`
`ch08_fattree_ecmp_restconf_groups_20260520_1009`	ECMP limitado mediante RESTCONF	Validada	Instalación de grupos *select* y flujos mediante RESTCONF con tráfico funcional.	`runs/ch08_fattree/`

Continúa en la página siguiente

Tabla C.1 – Continuación

Campaña o bloque	Tipo	Estado	Evidencia principal	Ruta base
`ch08_fattree_ecmp_full_20260521_0853`	Intento exploratorio en *datapath*	Exploratoria	*Multipath* en borde y uso de grupos, sin evidencia completa en agregación/*core*.	`runs/ch08_fattree/`
`ch08_fattree_ecmp_full_restconf_20260521_1705`	Intento exploratorio RESTCONF	Exploratoria	Migración RESTCONF viable, tráfico estable y evidencia parcial de *multipath*.	`runs/ch08_fattree/`
Procesado final del intento exploratorio	Procesado de resultados	Procesada	Tablas y figuras del intento exploratorio hacia ECMP completo.	`data/processed/ch08_fattree/`

C.2. Rutas de conservación de artefactos

La Tabla C.2 muestra las rutas principales donde se almacenan los artefactos derivados de las campañas. Estas rutas permiten localizar las evidencias utilizadas para construir las tablas, figuras y listados incluidos en la memoria.

Tabla C.2: Rutas principales de conservación de artefactos

Ruta	Contenido	Finalidad
`runs/ch08_fattree/`	Salidas brutas de campañas.	Comandos Mininet, JSON de `iperf3`, *dumps* OVS, resúmenes y evidencias por bloque.
`data/processed/ch08_fattree/`	Datos procesados del Capítulo 8.	Manifiestos, tablas intermedias, CSV consolidados y salidas normalizadas.
`figures/ch08_resultados/`	Figuras de resultados experimentales.	Gráficas de RTT, *throughput*, reparto de ramas y activación de *buckets*.
`tables/ch08/`	Tablas LaTeX del Capítulo 8.	Resúmenes de campañas, métricas, instalación RESTCONF y síntesis comparativas.
`listings/ch08/`	Listados del Capítulo 8.	Extractos de comandos, salidas de validación y secuencias reproducibles.
`tables/ch09/`	Tablas de discusión.	Síntesis crítica, limitaciones, amenazas de validez y lecciones aprendidas.
`tables/ch10/`	Tablas de conclusiones.	Cumplimiento de objetivos, trazabilidad frente a propuesta, síntesis final y trabajo futuro.
`data/processed/appendices/`	Manifiestos de apéndices.	Registro de generación de las tablas de apoyo de reproducibilidad.

C.3. Criterio de conservación

Durante el desarrollo del proyecto se conservó una separación entre evidencias brutas y evidencias procesadas. Las salidas brutas permanecen asociadas a cada campaña dentro de `runs/`, mientras que los resultados consolidados se almacenan en `data/processed/`, `figures/`, `tables/` y `listings/`. Esta organización evita sobrescribir resultados previos y permite reconstruir el origen de cada artefacto incluido en la memoria.

El criterio seguido en el documento ha sido integrar en los capítulos principales únicamente las evidencias necesarias para la argumentación técnica. Las rutas adicionales se conservan en este apéndice como apoyo de reproducibilidad y trazabilidad, sin convertir la memoria en un volcado completo del repositorio experimental.

BIBLIOGRAFÍA

[1] M. Al-Fares, A. Loukissas y A. Vahdat, «A Scalable, Commodity Data Center Network Architecture,» en *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ép. SIGCOMM '08, ACM, 2008, págs. 63-74. DOI: 10.1145/1402946.1402967. dirección: https://doi.org/10.1145/1402946.1402967.

[2] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya y A. Vahdat, «PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric,» en *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ép. SIGCOMM '09, ACM, 2009, págs. 39-50. DOI: 10.1145/1594977.1592575. dirección: https://doi.org/10.1145/1594977.1592575.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel y S. Sengupta, «VL2: A Scalable and Flexible Data Center Network,» en *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ép. SIGCOMM '09, ACM, 2009, págs. 51-62. DOI: 10.1145/1594977.1592576. dirección: https://doi.org/10.1145/1594977.1592576.

[4] P. J. Roig, S. Filiposka, S. Alcaraz, K. Gilly y N. Aknin, «Formal Algebraic Specification of an IoT/Fog Data Centre for Fat Tree or Leaf and Spine Architectures,» en *2020 2nd International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, IEEE, 2020, págs. 1-6. DOI: 10.1109/ICECCE49384.2020.9179445. dirección: https://doi.org/10.1109/ICECCE49384.2020.9179445.

[5] K. C. Okafor, I. E. Achumba, G. A. Chukwudebe y G. C. Ononiwu, «Leveraging Fog Computing for Scalable IoT Datacenter Using Spine-Leaf Network Topology,» *Journal of Electrical and Computer Engineering*, vol. 2017, págs. 1-11, 2017. DOI: 10.1155/2017/2363240. dirección: https://doi.org/10.1155/2017/2363240.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker y J. Turner, «OpenFlow: Enabling Innovation in Campus Networks,» *ACM SIGCOMM Computer Communication Review*, vol. 38, n.º 2, págs. 69-74, 2008. DOI: 10.1145/1355734.1355746. dirección: https://doi.org/10.1145/1355734.1355746.

[7] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky y S. Uhlig, «Software-Defined Networking: A Comprehensive Survey,» *Proceedings of*

the IEEE, vol. 103, n.º 1, págs. 14-76, 2015. DOI: 10.1109/JPROC.2014.2371999. dirección: https://doi.org/10.1109/JPROC.2014.2371999.

[8] E. Haleplidis, K. Pentikousis, S. Denazis, J. Hadi Salim, D. Meyer y O. Koufopavlou, «Software-Defined Networking (SDN): Layers and Architecture Terminology,» Internet Engineering Task Force, RFC 7426, ene. de 2015. DOI: 10.17487/RFC7426. dirección: https://www.rfc-editor.org/rfc/rfc7426.html.

[9] Universidad Miguel Hernández de Elche, «Propuesta de investigación: Implementación SDN de arquitecturas Fat Tree multidimensionales,» Documento interno de propuesta del Trabajo Fin de Grado, 2024.

[10] B. Lantz, B. Heller y N. McKeown, «A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,» en *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ép. HotNets-IX, Monterey, CA, USA: ACM, 2010. DOI: 10.1145/1868447.1868466. dirección: https://doi.org/10.1145/1868447.1868466.

[11] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon y M. Casado, «The Design and Implementation of Open vSwitch,» en *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*, ép. NSDI 15, USENIX Association, 2015, págs. 117-130. dirección: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff.

[12] Open Networking Foundation, *OpenFlow Switch Specification, Version 1.3.0*, Open Networking Foundation, jun. de 2012. visitado 25 de mayo de 2026. dirección: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

[13] OpenDaylight Project, *OpenDaylight Documentation: Titanium Downloads and Getting Started Guide*, OpenDaylight, 2026. visitado 25 de mayo de 2026. dirección: https://docs.opendaylight.org/en/stable-titanium/downloads.html.

[14] Open vSwitch Project, *ovs-ofctl: Administer OpenFlow Switches*, Open vSwitch, 2026. visitado 25 de mayo de 2026. dirección: https://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt.

[15] OpenDaylight Project, *OpenDaylight OpenFlow Plugin: Operation and RESTCONF APIs*, OpenDaylight, 2026. visitado 25 de mayo de 2026. dirección: https://docs.opendaylight.org/projects/openflowplugin/en/latest/users/operation.html.

[16] A. Bierman, M. Bjorklund y K. Watsen, «RESTCONF Protocol,» Internet Engineering Task Force, RFC 8040, ene. de 2017. DOI: 10.17487/RFC8040. dirección: https://www.rfc-editor.org/rfc/rfc8040.html.

[17] C. E. Hopps, «Analysis of an Equal-Cost Multi-Path Algorithm,» Internet Engineering Task Force, RFC 2992, nov. de 2000. DOI: 10.17487/RFC2992. dirección: https://www.rfc-editor.org/rfc/rfc2992.html.

[18] Open vSwitch Project, *Open vSwitch Documentation: OpenFlow FAQ*, Open vSwitch, 2026. visitado 25 de mayo de 2026. dirección: https://docs.openvswitch.org/en/latest/faq/openflow/.

[19] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang y A. Vahdat, «Hedera: Dynamic Flow Scheduling for Data Center Networks,» en *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, ép. NSDI 10, San Jose, CA, USA: USENIX Association, 2010. dirección: https://www.usenix.org/conference/nsdi10-0/hedera-dynamic-flow-scheduling-data-center-networks.

[20] Mininet Project, *Mininet: An Instant Virtual Network on your Laptop or other PC*, Mininet Project, 2026. visitado 25 de mayo de 2026. dirección: https://github.com/mininet/mininet.

[21] T. Ambrazavicius, «Implementación SDN de Arquitecturas Spine & Leaf Multidimensionales,» Directores: Salvador Alcaraz Carrasco y Pedro Juan Roig Roig, Trabajo Fin de Grado, Universidad Miguel Hernández de Elche, dic. de 2024.