Universidad de Alcalá

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Nuevos conmutadores de red para redes integradas con SDN

Tesis Doctoral presentada por

Diego López Pajares

2021

Universidad de Alcalá

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Nuevos conmutadores de red para redes integradas con SDN

Tesis Doctoral presentada por

Diego López Pajares

Directores

Juan Antonio Carral Pelayo y Elisa Rojas Sánchez

Alcalá de Henares, abril de 2021

"La mente que se abre a una nueva idea jamás volverá a su tamaño original"
Albert Einstein

Este trabajo ha sido parcialmente financiado por la Universidad de Alcalá a través de los proyectos BISHOPS (CCG2018/EXP-076), SIMPSONS (CCG2017/EXP-001) y ESPECIE (UAH INFR.B 2017-007); por la Comunidad de Madrid a través de los proyectos TIGRE5-CM (S2013/ICE-2919), TAPIR-CM (S2018/TCS-4496) e IRIS (CM/JIN/2019-039); y por la Junta de Comunidades de Castilla la Mancha a través del proyecto IRIS (SBPLY/19/180501/000324).

Agradecimientos

La gratitud da sentido a nuestro pasado, trae paz al presente y crea una visión para mañana.

Anónimo.

Por fin llega el momento de escribir el apartado más reconfortante de la Tesis, los agradecimientos. Esta aventura comenzó por casualidad, cuando aún era estudiante del Máster Universitario en Ingeniería de Telecomunicación y empecé a buscar ofertas de trabajo dentro de la universidad que me permitiesen compatibilizar mis estudios de máster con un trabajo remunerado. Entre las ofertas a las que apliqué se encontraba la del grupo Networks and Intelligent Systems (NetIS), que apostaron por contratarme a mí entre otros candidatos al puesto. Aquel momento cambió el rumbo de mi vida, cuyas miras de futuro estaban enfocadas por aquel entonces al mundo empresarial y, ahora, se centran en el mundo académico y de investigación. Es de recibo agradecer aquella oportunidad, pues abrió en mí un mundo totalmente desconocido, en el que ahora me encuentro completamente integrado y feliz. De aquella relación laboral surgió algo más que eso, una "pequeña familia" a la que perteneceré siempre.

También quiero agradecer a mis directores de Tesis su esfuerzo, apoyo y dedicación. Sin sus consejos está Tesis no sería posible, ya que han hecho de guía en momentos coyunturales y han sabido motivarme para continuar en esta aventura.

Aprovecho para agradecer también esos mensajes de apoyo constantes recibidos en esta última etapa que me han ayudado a dar el último empujón para acabar la Tesis.

Por último, también agradezco a mi familia su apoyo tanto psicológico y emocional como económico a lo largo de los años, que ha hecho posible que hoy me encuentre escribiendo estas líneas. Sin lugar a dudas, esto, sin ellos, no habría sido posible.

A todos, muchísimas gracias.

Resumen

La revolución tecnológica en la que estamos inmersos es proclive a producir cambios de paradigma total o parcialmente incompatibles con modelos o arquitecturas previas, como supuso, por ejemplo, el nacimiento de las redes Software-Defined Networking (SDN). En concreto, la tecnología SDN ofrece un modelo de red centralizado, en el que un único elemento lógico, el controlador, es el responsable de gestionar y mantener la red, lo que choca con el paradigma anterior, un modelo de red totalmente distribuido en el que primaba la cooperación entre dispositivos para la gestión y el mantenimiento de la red. Precisamente, la gestión centralizada de SDN permite realizar un despliegue de servicios a alto nivel de forma dinámica y escalable, lo que favorece al rápido crecimiento de las redes de comunicaciones. Por este motivo, el concepto de red SDN está en auge, y forma parte del núcleo de la red en las redes de nueva generación 5G y 6G. Sin embargo, al ser una tecnología relativamente reciente, aún tiene retos por resolver.

Esta Tesis aprovecha el reto del multicamino para resolver desafíos aún presentes en las redes SDN como, por ejemplo, la falta de estandarización en redes SDN In-Band, o la reducción de carga computacional en el controlador de la red. Para enmendar dichas cuestiones, se introducen técnicas que combinan modelos de red centralizados y distribuidos, como son las redes SDN híbridas, que delegan ciertos servicios de red, dependientes del controlador, a los switches de la red para que los ejecuten de forma distribuida. Además, para afrontar dichos desafíos, se utilizan técnicas de exploración de red que, mediante una difusión controlada de paquetes, generan información topológica útil para construir posteriormente los caminos, en lugar de emplear técnicas basadas en el estado del enlace, que construyen los caminos mediante un intercambio activo de información entre switches vecinos para notificar un cambios estado o mejoras del coste de los enlaces. Por último, se aprovechan los conocimientos adquiridos en el reto del multicamino en redes SDN para diseñar un nuevo algoritmo de búsqueda de múltiples caminos disjuntos "genérico", aplicable tanto a redes de comunicaciones como a otras disciplinas que no guardan relación con las redes de datos.

Palabras clave: SDN, Múltiples caminos, Caminos disjuntos, Redes híbridas, Algoritmos.

Abstract

The technological revolution in which we are immersed tends to create new paradigms that are totally or partially inconsistent with previous models or architectures, as implied, for example, the birth of SDN networks. Specifically, SDN technology offers a centralised network model with a logical device, the controller, which is responsible for managing and maintaining the network. The aforementioned model is incompatible with the previous network paradigm, a distributed network model that relies on the cooperation of devices for network management. Precisely, centralized network management allows a dynamic and scalable deployment of high-level services that stimulate the fast-growth of SDN networks. For these reasons, the SDN concept is on the rise and is the basis on which new generation networks (5G and 6G) are built.

This Ph.D. leverages the multipath challenge to solve challenges still present in SDN networks, such as the lack of standardisation in SDN In-Band network model, or the computational load reduction in the controller. To solve them, a new concept is included, the hybrid-SDN network, in which the controller delegates some functions to network switches to work in a distributed way. Moreover, both challenges introduce network exploration techniques to generate useful topological information to find multiple paths, instead of using link-state protocols to exchange such topological information among neighbors. Finally, the previous knowledge of hybrid-SDN networks is taken into account to create a new versatile multipath search algorithm, that is applicable both to communication networks and to other non-network-related disciplines.

Keywords: SDN, Multipath, Disjoint paths, Hybrid-networks, Algorithms.

Índice general

Índice de figuras

Índice de tablas

Índice de Algoritmos

Lista de acrónimos

1S-MDP	One Shot Multiple Disjoint Paths.
AMAC	Amaru MAC.
API	Application Programming Interface.
ARoot	Amaru Root.
ARP	Address Resolution Protocol.
BFS	Breadth First Search.
BOFUSS	The basic OpenFlow userspace software switch.
BRITE	Boston university Representative Topology gEnerator.
CapEx	Capital Expenditure.
DFS	Deep First Search.
EBID	Estructura Básica de Información por Dispositivo.
gRPC	google Remote Procedure Call.
I-MDP	Iterative Multiple Disjoint Paths.
IAT	Inter Arrival Time.
IGP	Interior Gateway Protocol.
ISP	Internet Services Provider.
MAC	Media Access Control.
MAN	Metropolitan Area Network.
MDPAlg	Multiple Disjoint Paths Algorithm.
ML	Machine Learning.
MPTCP	MultiPath TCP.

NDP	Neighbour Discover Protocol.
NetIS	Networks and Intelligent Systems.
NFV	Network Function Virtualization.
ONF	Open Networking Foundation.
ONOS	Open Network Operating System.
OpEx	Operating Expense.
OSI	Open System Interconnection.
OSPF	Open Shortest Path First.
OVS	Open Virtual Switch.
QoS	Quality of Service.
RSTP	Rapid Spanning Tree Protocol.
RTT	Round-Trip Time.
SDN	Software-Defined Networking.
SPB	Shortest Path Bridging.
STP	Spanning Tree Protocol.
TIC	Tecnologías de la Infomación y las Comunicaciones.
TRILL	Transparent Interconnection of Lots of Links.
VLAN	Virtual Local Area Network.
WAN	Wide Area Network.

Capítulo 1

Introducción y objetivos

Este primer capítulo presenta el marco general de la Tesis, describe los objetivos a alcanzar, define la estructura general del documento, y resume las contribuciones científicas que ha aportado esta Tesis.

1.1 Introducción

Las redes definidas por software, del inglés Software-Defined Networking (SDN) [Open Networking Fundation, 2012], ofrecen un modelo de arquitectura de red programable a alto nivel que abstrae al administrador de la red de los comandos de configuración de bajo nivel que manejan los switches. Tal objetivo se consigue centrando la inteligencia en la red en un único dispositivo, el controlador, quien se encarga de mantener y gestionar la red al actuar de interlocultor entre las órdenes de alto nivel utilizadas por el administrador, y las órdenes de bajo nivel manejadas por los switches. Estas facetas han propiciado la rápida adopción de las redes SDN debido, principalmente, a su capacidad de adaptación y a la rapidez en el despliegue de nuevos servicios, cualidades demandadas por la sociedad actual, la cual, está permanentemente conectada a Internet y exige cada vez más servicios de red. Además, esta sociedad necesita progresivamente más ancho de banda, por lo que surge la necesidad de utilizar varias conexiones en paralelo (múltiples rutas) para aumentar las tasas de transferencia de datos.

El uso de múltiples rutas en redes de comunicaciones, además de incrementar la tasa de transferencia de datos, también tiene otras posibles aplicaciones como, por ejemplo, aumentar la fiabilidad de la red enviando los datos duplicados a través de varios caminos, o incrementar la seguridad de los datos dividiendo la información transmitida entre varios caminos, evitando así ataques del tipo *Man in the middle*. De ahí surge la idea de combinar la tecnología SDN y el uso de caminos múltiples, dando lugar a esta Tesis Doctoral, centrada en diseñar protocolos que buscan múltiples caminos en redes SDN.

1.2 Redes SDN

Las redes SDN junto con la tecnología de virtualización de funciones de red, del inglés Network Function Virtualization (NFV) [ETSI, 2012], han transformado radicalmente la industria de las redes de comunicaciones gracias al despliegue, mantenimiento y gestión de servicios de red de forma dinámica y escalable, permitiendo al administrador de red operar a alto nivel desde un único punto lógico de control. Este tipo de tecnologías suponen un cambio de paradigma en las arquitecturas de red y la industria de las redes de datos actuales, ya que se parte de un modelo distribuido con hardware y software propietario de empresas como Cisco, Juniper, Avaya, HP, NEC, y organismos de estandarización clásicos (IETF, IEEE), a un modelo con un control centralizado utilizado por grandes empresas tecnológicas como Google [Vahdat et al., 2015] o Telefónica [Montero et al., 2017] y certificado por la Open Networking Foundation (ONF) [Open Networking Foundation, 2020], empleando además hardware y software libre. El apoyo de grandes empresas tecnológicas al desarrollo y despliegue de este tipo de tecnologías viene motivado por los beneficios que aporta al sector de las Tecnologías de la Infomación y las Comunicaciones (TIC), ya que la re-utilización de hardware para desplegar nuevos servicios y aplicaciones permite reducir el gasto en capital, del inglés Capital Expenditure (CapEx), y la facilidad de operación y gestión mediante la programabilidad de equipos reduce el gasto de operación, del inglés Operating Expense (OpEx). Además, se incrementa la flexibilidad y agilidad de operación mediante el despliegue de nuevas aplicaciones y servicios de una forma rápida y eficaz, y se incentiva la investigación gracias a su potencial de crecimiento. También han contribuido al rápido despliegue de las redes SDN los cambios en los hábitos de vida en la última década. Los usuarios han incrementado el volumen del tráfico de datos por la red gracias al consumo de contenidos digitales, el uso de las redes sociales, los sistemas de computación en la nube, la realización de videollamadas y teleconferencias, la apuesta por la sede electrónica, etc. Este aumento del volumen de tráfico complica la gestión manual de la red por parte de sus administradores, alejando la idea de una gestión y mantenimiento manual y apostando fuertemente por modelos de red programables como son las redes SDN. Otra característica fundamental de este tipo de redes es su capacidad para recoger estadísticas en vivo del estado de la red (número de paquetes transmitidos, número de paquetes erróneos, interfaces habilitadas, caídas de interfaces, etc.), concentrando todos los datos en un único punto de control. Estos estadísticos son clave para introducir técnicas de Machine Learning (ML) que permitan a la red tomar decisiones de forma autónoma y aprender patrones de comportamiento que le permitan tomar acciones previas ante situaciones de riesgo, lo que incrementa aún más los beneficios de las redes SDN.

La arquitectura SDN desacopla el plano de control (mantenimiento, gestión y control de la red) del plano de datos (lógica de reenvío), funciones tradicionalmente integradas en una única capa lógica. Este desacoplo aísla los mensajes de control de la red, de los

datos de usuario, incrementando la seguridad, la operabilidad y la manejabilidad de la red. Además, dicha división lógica tiene una relación directa con la infraestructura física (hardware), ya que el plano de datos recae sobre equipos de conmutación de paquetes, mientras que el plano de control está integrado en un nuevo dispositivo denominado controlador. Con esta separación de planos se garantiza la escalabilidad y fiabilidad de equipos, ya que un único controlador puede gestionar varios conmutadores. No obstante, también se pueden añadir varios controladores a una red SDN como sistemas de respaldo en caso que el controlador principal falle, o para para sementar la red en áreas de menor tamaño (reducir el número de conmutadores por controlador).

En cuanto a la conexión física entre el controlador y los conmutadores, existen dos alternativas: modelo Out-of-Band y modelo In-Band. En el modelo Out-of-Band cada conmutador cuenta con una conexión dedicada y directa con el controlador que aísla totalmente los datos de control y los datos de usuario, mientras que en el modelo In-Band solamente un conmutador está conectado directamente con el controlador, y la comunicación del control con el resto de conmutadores se realiza a través de las conexiones físicas existentes entre dichos conmutadores. En este último modelo, el tráfico de control y el tráfico de usuario están separados solamente de forma "lógica" ya que comparten los mismos recursos físicos, ahora bien, el tráfico de control tiene prioridad sobre el tráfico de datos para garantizar la operabilidad de la red. Sin embargo, el modelo In-Band no cuenta con una estandarización oficial como sí tiene el modelo Out-of-Band, a pesar de que el modelo In-Band hace un mejor aprovechamiento de los recursos físicos que el modelo Out-of-Band. Es, precisamente, esta falta de estandarización, la responsable de abrir una línea de investigación en la que se basa una de las aportaciones de la Tesis, a la que además se le suma el reto de la multiplicidad de caminos.

1.3 Múltiples caminos en redes de comunicaciones

Los protocolos de encaminamiento surgieron con el objetivo de distribuir información entre usuarios ubicados en distintos puntos de la red, encargándose de buscar y establecer rutas que comuniquen dichos dispositivos. Tradicionalmente, en las redes de conmutadores (switches), se utilizan protocolos como Spanning Tree Protocol (STP) [IEEE, 1998] o Rapid Spanning Tree Protocol (RSTP) [IEEE, 2001], que eliminan caminos redundantes para evitar las tormentas de mensajes provocadas por los paquetes de broadcast. El resultado es un árbol de expansión enraizado en un switch que llega al resto de switches de la red y está libre de bucles. Esta solución, si bien es efectiva para anular las tormentas de paquetes, no es óptima en el uso de recursos, ya que únicamente se están utilizando un subconjunto de enlaces de la red. Además, si hay tráfico continuo y constante entre varios pares de switches, los enlaces del árbol tienden a saturarse, lastrando la experiencia de usuario. Por este motivo surgieron alternativas como Shortest Path Bridging (SPB) [IEEE, 2012] o Transparent Interconnection of Lots of Links (TRILL) [Perlman & Eastlake,

2011], protocolos que utilizan la técnica *link-state*, en la que los switches intercambian información sobre el coste de los enlaces con el objetivo de obtener las rutas de mínimo coste o menor longitud entre dos pares de switches. Estas propuestas mejoran a sus predecesoras al aprovechar por completo la red y poder descubrir múltiples caminos entre cada par de switches. Es aquí donde entra en juego la multiplicidad de caminos y sus diferentes aplicaciones. Al disponer de varios caminos, estos se pueden utilizar en paralelo para incrementar el ancho de banda de un flujo de datos, para asegurar la integridad de los datos transmitidos al segmentar un flujo entre varios caminos, o para tener caminos de respaldo en caso de fallo, aplicaciones que no son posibles con una solución monocamino. Además, estas cualidades se ven reforzadas si los múltiples caminos son disjuntos entre sí (no comparten recursos físiscos a excepción de los switches extremos), puesto que lo que ocurre en un camino no afecta al resto. Los múltiples caminos disjuntos se pueden clasificar en dos categorías atendiendo a su estructura:

- Caminos disjuntos en enlaces: Son aquellos caminos que no comparten enlaces entre sí.
- Caminos disjuntos en nodos: Son aquellos caminos que no comparten ningún nodo ni ningún enlace entre sí, a excepción de los nodos extremos.

En el caso particular de las redes de switches cableadas, existe una correspondencia directa entre nodos y switches, y entre los enlaces y las conexiones físicas que unen los switches. Los caminos disjuntos en enlaces ofrecen un aislamiento parcial, ya que varios caminos pueden compartir, además de los nodos extremos, otros nodos intermedios siempre que se garantice que sus enlaces son únicos. No obstante, los caminos disjuntos en nodos sí que garantizan ese aislamiento total entre caminos, por lo que aportan mayores beneficios. Sin embargo, en este último caso se obtiene un menor número de caminos disjuntos debido a las restricciones más duras que impone su construcción.

Las soluciones multicamino (con o sin disjuntividad) pueden integrarse perfectamente en las redes SDN de nueva generación, siendo el controlador el encargado de descubrir los múltiples caminos. Para ello, en primer lugar, el controlador recoge información sobre la infraestructura de red subyacente y la representa en forma de grafo, y después aplica un algoritmo sobre dicho grafo que calcula los múltiples caminos existentes entre los switches de la red. Sin embargo, este enfoque presenta un problema de escalabilidad, ya que el tiempo de resolución del problema se incrementa de forma exponencial conforme aumenta el tamaño de la red. Por lo tanto, descubrir múltiples caminos en redes puede llevar al controlador un tiempo no despreciable, crucial para resolver otros problemas de mayor importancia como atender calidades de servicio o monitorizar y detectar problemas en la red. Además, este efecto se puede ver incrementado si los caminos que se desean obtener son disjuntos, puesto que su cálculo requiere un mayor esfuerzo matemático. La idea que se persigue consiste en desplazar el cálculo de caminos del controlador a la capa

de infraestructura (los switches de la red), para que sea ésta la que calcule los caminos de forma distribuida y devuelva el resultado obtenido al controlador. De esta forma se reduce la carga computacional del controlador, liberándolo para atender a otras tareas de mayor prioridad, al mismo tiempo que el controlador conoce los caminos disjuntos descubiertos para poder utilizarlos en los servicios que lo necesiten. Para resolver este reto hay que "hibridizar" la red SDN, haciendo que la capa de infraestructura tome el control de ciertas funciones de red, creando un entorno mixto entre el enfoque distribuido que ofrecen las redes tradicionales, y la visión centralizada de las redes SDN. Además, como valor añadido, se pretende que los caminos obtenidos de esta forma sean bien disjuntos en nodos, bien disjuntos en enlaces.

1.4 Planteamiento del problema y definición de objetivos

El objetivo propuesto en esta Tesis Doctoral consiste en diseñar nuevos protocolos de red multicamino que resuelvan limitaciones presentes en las redes SDN como, por ejemplo, la falta de una solución robusta y estandarizada en redes SDN In-Band, o la sobrecarga computacional a la que están sometidos los controladores SDN.

En relación a las redes SDN In-Band, su despliegue supone un ahorro de costes con respecto al modelo Out-of-Band, al reutilizar la infraestructura existente en la capa de infraestructura para transmitir tanto el tráfico de control como el tráfico de datos. Por lo tanto, el concepto de red SDN In-Band resulta interesante para el despliegue de redes SDN que abarcan grandes superficies de terreno, evitando la instalación de una red paralela dedicada exclusivamente al tráfico de control. Como ejemplo de redes que abarcan grandes extensiones de terreno se encuentran las redes Wide Area Network (WAN), o redes pertenecientes a Internet Services Provider (ISP) cuya superficie cubre millones de kilómetros cuadrados como, por ejemplo, la red Pan-Europea [De Maesschalck et al., 2003], que conecta países distribuidos por el territorio europeo, o la red de fibra Rogers Fiber Backbone [Rogers Communications Inc., 2018], que recorre de este a oeste Estados Unidos y Canadá. A pesar de las ventajas que ofrece el enfoque SDN In-Band, éste no cuenta con un mecanismo estandarizado que regule la comunicación entre el controlador y los switches de la red a través de la capa de infraestructura, por lo que se presenta un nicho de investigación ideal para diseñar un protocolo que comunique el controlador con los switches en una red SDN In-Band. Además, este escenario es ideal para incluir multiplicidad de caminos, desarrollando un protocolo que proporcione múltiples caminos con el objetivo de aumentar la fiabilidad del sistema y minimizar el tiempo de recuperación ante caídas parciales de la red.

Por otro lado, la sobrecarga de cómputo a la que están sometidos los controladores en la actualidad incrementa los requisitos hardware mínimos exigibles para ejecutar todo el software de gestión y mantenimiento de la red, lo cual restringe una de las características

fundamentales de las redes SDN, su escalabilidad. Por ese motivo, uno de los problemas a tratar por la presente Tesis Doctoral, es la búsqueda de procedimientos que obtengan múltiples caminos disjuntos reduciendo la carga de cómputo del controlador. Es aquí donde se buscarán alternativas a los enfoques de red existentes en la actualidad, mezclando las bondades de las redes tradicionales con los beneficios de las redes SDN, ofreciendo soluciones que obtengan las múltiples rutas con la intervención pasiva del controlador. La propuesta pasa por un modelo de red híbrido, en el que los dispositivos de red calculen las múltiples rutas de forma descentralizada para, posteriormente enviarle las rutas descubiertas al controlador. De esta forma, el controlador solo tiene que almacenar la ruta previamente calculada para su posterior uso, eliminando el proceso de descubrimiento topológico y el posterior cálculo de rutas necesario en un enfoque SDN clásico.

No obstante, no siempre es posible trabajar en entornos SDN híbridos, bien porque el hardware de la capa de infraestructura no admite modificaciones, bien por que interfiera con otros propósitos. Por lo tanto, se pretende aprovechar los conocimientos adquiridos en redes SDN híbridas para diseñar un algoritmo de búsqueda de caminos disjuntos que sea eficiente computacionalmente. De esta forma, se puede reducir la complejidad computacional del controlador incluso cuando no sea posible utilizar un entorno SDN híbrido, ya que el controlador puede ejecutar este algoritmo de forma centralizada. Además, este algoritmo abre las puertas a nuevos nichos de investigación alejados de las redes de comunicaciones que necesiten utilizar múltiples caminos disjuntos.

Por último, las siguientes líneas recogen, a modo de resumen, los principales objetivos de la Tesis:

- El diseño de un protocolo que comunique el plano de control a través de la capa de infraestructura en las redes SDN In-Band. Este protocolo debe incluir en la solución multiplicidad de caminos para aumentar la fiabilidad y robustez de la propuesta ante fallos o caídas en la red.

- El diseño de propuestas multicamino que reduzcan la carga computacional del controlador de la red utilizando un enfoque de red SDN híbrido. Además, estos caminos deben ser disjuntos por las ventajas adicionales que aportan y el sobreesfuerzo de cómputo que supone su cálculo para el controlador.

- El desarrollo de un algoritmo de búsqueda de caminos disjuntos que trabaje de forma centralizada, partiendo de los conocimientos adquiridos en el diseño de protocolos en redes SDN híbridas. Este diseño debe ser agnóstico a la aplicación final, para que el algoritmo sea válido tanto en entornos de red como en otras disciplinas.

1.5 Estructura de la memoria

A continuación, se describe la estructura de la memoria y el contenido de sus capítulos, recogiendo en un único apartado el esquema general del documento, con el objetivo de proporcionar una visión general del marco de la Tesis que facilite su lectura.

El primer capítulo ha introducido el contexto general de las redes SDN, así como los beneficios que aportan los múltiples caminos en las redes de comunicaciones, los dos temas principales sobre los que trata la Tesis. Además, se ha planteado brevemente el problema y se han expuesto los objetivos que se pretenden alcanzar, finalizando el capítulo con un resumen de las aportaciones científicas que ha logrado esta Tesis.

El segundo capítulo recoge el estado del arte relacionado con la temática de la tesis, dividiendo el capítulo en dos grandes bloques, las redes SDN y el problema de los múltiples caminos en las redes de comunicaciones. El primero de ellos comienza con una descripción de la arquitectura de red SDN, para después profundizar en el modelo de comunicación In-Band, y acabar con un modelo híbrido de red SDN que combina las ventajas de un entorno centralizado con los beneficios de los sistemas distribuidos. El segundo bloque describe el problema de los múltiples caminos en redes de comunicaciones, mostrando la terminología básica y clasificando los trabajos en función del tipo de caminos obtenidos (múltiples caminos, o múltiples caminos disjuntos).

El tercer capítulo plantea con mayor nivel de detalle los problemas a resolver y los puntos de actuación, sirviendo además de introducción a los siguientes capítulos, en los que se detallan las contribuciones de la Tesis.

El cuarto capítulo está directamente relacionado con el primer objetivo de la Tesis y muestra Amaru, un protocolo orientado al descubrimiento de múltiples caminos en redes SDN In-Band que encaminen el tráfico de control de la red SDN a través de la capa de infraestructura, y restauren fallos de red de forma rápida. El capítulo incluye una descripción detallada del proceso de descubrimiento llevado a cabo por Amaru, del posterior procedimiento de comunicación entre el controlador y los switches de la red, y del mecanismo de restauración de fallos implementado por Amaru. Además, se incluyen apartados específicos que describen la implementación de Amaru y los resultados obtenidos en el proceso de evaluación.

El quinto capítulo, alineado con el segundo objetivo, se centra en dos protocolos, Iterative Multiple Disjoint Paths (I-MDP) y One Shot Multiple Disjoint Paths (1S-MDP), cuyo objetivo es la búsqueda de caminos disjuntos en redes SDN híbridas. Ambos protocolos realizan una búsqueda de caminos sin la intervención activa del controlador, delegando la búsqueda de caminos a los dispositivos de la capa de infraestructura. 1S-MDP es una versión mejorada I-MDP que realiza la búsqueda de caminos más eficientemente. Al igual que el capítulo anterior, éste incluye detalles sobre la operativa de los protocolos y el proceso llevado a cabo para su implementación, junto con un apartado dedicado al

proceso de evaluación.

El sexto capítulo aprovecha los conocimientos adquiridos en el capítulo anterior para transformar 1S-MDP en un algoritmo que trabaja de forma centralizada, Multiple Disjoint Paths Algorithm (MDPAlg), de acuerdo con el tercer y último objetivo de la Tesis. Este nuevo enfoque se aleja ligeramente de las redes de comunicaciones y sienta las bases para iniciar una nueva línea de investigación orientada al mundo de la teoría de grafos. El capítulo incluye una descripción detallada del algoritmo, así como un estudio teórico de la complejidad computacional que presenta MDPAlg, concluyendo con una sección de evaluación que valida el estudio teórico y cuantifica tanto el número de caminos disjuntos descubiertos por MDPAlg como el tiempo que invierte en conseguirlos.

Por último, el séptimo capítulo recoge las conclusiones principales de la Tesis, así como un apartado que describe futuras líneas de investigación.

1.6 Contribuciones

El trabajo realizado en esta Tesis Doctoral ha aportado a la comunidad científica dos trabajos publicados en revistas indexadas en el índice JCR con alto impacto (1 Q1 y 1 Q2), y un tercero en proceso de revisión (Q1), una ponencia de conferencia calificada con un nivel A en el ranking Core, y dos patentes nacionales, una aceptada y otra que está solicitada y ha superado la fase de búsqueda. A continuación se exponen, a modo de resumen, todas las contribuciones que ha aportado esta Tesis a la comunidad científica.

Artículos de revista indexadas de alto impacto:

1. D. Lopez-Pajares, J. Alvarez-Horcajo, E. Rojas, A. S. M.Asadujjaman, & I. Martinez-Yelmo. (2019). Amaru: Plug&play resilient in-band control for SDN. IEEE Access, 7, 123202-23218. (JCR Q1) [Lopez-Pajares et al., 2019].

2. D. Lopez-Pajares, J. Alvarez-Horcajo, E. Rojas, J. A. Carral & I. Martinez-Yelmo. (2020). One-Shot Multiple Disjoint Path Discovery Protocol (1S-MDP). IEEE Communications Letters, 24(8), 1660-1663. (JCR Q2) [Lopez-Pajares et al., 2020].

3. D. Lopez-Pajares, E. Rojas, J. A. Carral, I. Martinez-Yelmo & J. Alvarez-Horcajo. The disjoint multipath challenge: multiple disjoint paths guaranteeing scalability, IEEE Access, under review. (JCR Q1).

Ponencia en conferencia internacional:

1. D. Lopez-Pajares, J. Alvarez-Horcajo, E. Rojas, J. A. Carral & G. Ibanez. (2018). Iterative Discovery of Multiple Disjoint Paths in Switched Networks with Multicast Frames. IEEE 43rd Conference on Local Computer Networks (LCN), 409-412. (CORE A) [Lopez-Pajares et al., 2018].

Patentes:

1. G. Ibanez, J. Alvarez-Horcajo, J. A. Carral, I. Martinez-Yelmo, & D. Lopez-Pajares. Procedimiento de establecimiento y borrado de caminos mu?ltiples disjuntos, de reenvio de tramas y puente de red. Patente de invención. Nº de patente: ES2638292B2. Fecha de concesión 17/04/2018. [Ibanez et al., 2018].

2. E. Rojas, D. Lopez-Pajares, J. A. Carral, & J. Alvarez-Horcajo. Procedimiento de descubrimiento de caminos múltiples disjuntos en paso y nodo de red. Patente de invención. En proceso de solicitud. Nº de solicitud: P201931036. Fecha de solicitud 22/11/2019.

Capítulo 2

Estado del Arte

Este capítulo realiza una revisión del estado del arte recopilando información relacionada directa o indirectamente con la temática de esta Tesis.

Al principio, el capítulo se centra en el concepto de red SDN, describiendo su descomposición lógica y física, para más tarde hacer hincapié en el modelo In-Band y en el concepto de red SDN híbrida. Después de mostrar en detalle la composición de las redes SDN, el capítulo trata el problema del multicamino en las redes de comunicaciones definiendo, en primer lugar, la terminología básica, para después dividir el problema en diferentes categorías y mostrar los detalles de cada una de ellas.

2.1 Redes SDN

Las redes SDN introducen un concepto de arquitectura de red disruptor con la estructura arquitectónica previa, con un elemento central, el controlador, que permite gestionar y mantener la red de forma inteligente, "programándola" mediante aplicaciones software. La gestión y el mantenimiento centralizado simplifican al máximo el trabajo del administrador de la red, al poder conocer el estado de la red y definir su comportamiento desde un único punto. Por otro lado, el controlador es capaz de abstraer las órdenes de bajo nivel manejadas por los switches hardware en funciones simples con una sintaxis natural. Todas estas características proporcionan una red dinámica capaz de adaptarse rápidamente a los cambios, ya que el administrador del sistema puede actualizar el comportamiento del conjunto de la red con tan solo instalar una aplicación software en el controlador, lo que reduce los costes de operación, despliegue y mantenimiento. Además, las redes SDN hacen una apuesta firme por el software y hardware libre a fin de obtener una arquitectura de red global que elimine barreras de entrada y no dependa de soluciones propietarias.

2.1.1 Arquitectura de red lógica en redes SDN

SDN ofrece una división lógica de la red en dos planos, el plano de control y el plano de datos, y tres capas, aplicación, control e infraestructura, como se recoge en la Figura 2.1.

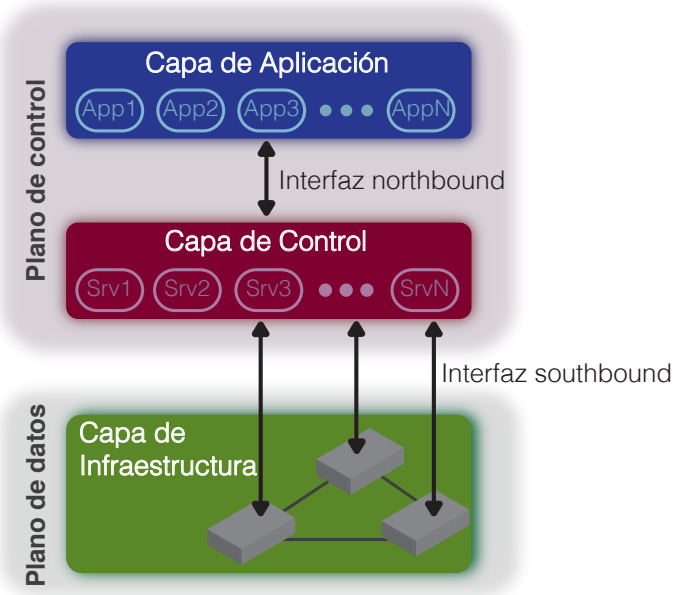


Figura 2.1: Arquitectura lógica de las redes SDN.

El plano de control (el controlador) está compuesto por las capas de aplicación y control, que se comunican a través de la interfaz norte, o interfaz northbound, mientras que el plano de datos lo compone la capa de infraestructura, compuesta por los dispositivos o recursos de red, también conocidos como switches SDN. Ambos planos (control y datos) se conectan a través de la interfaz sur, o interfaz southbound. En el plano de control reside el "cerebro" de la red, que se encarga de su gestión y mantenimiento. Entre sus funciones se encuentran tareas tan diversas como la configuración de rutas en los dispositivos de la capa de infraestructura, la supervisión del estado de la red, o la recolección de estadísticas. El plano de datos simplemente se encarga de ejecutar las órdenes impuestas por el plano de control, reenvío o descarte de paquetes, y envío de estadísticos al controlador. Por lo tanto, bajo esta visión lógica, quedan claramente definidos los roles de cada plano, la "inteligencia" de la red se queda en el plano de control, y la "mano ejecutora" en el plano de datos, descomposición que rompe con la lógica de las redes tradicionales, donde el plano de control y el plano de datos coexistían en el mismo dispositivo. Esta visión permite diferenciar claramente entre dispositivos de reenvío (switches) y el dispositivo de control (controlador), reduciendo los costes de despliegue y operación, al centrar en un único elemento la potencia de hardware y proveer al resto de dispositivos un hardware más enfocado al reenvío de paquetes. No obstante, para tener una visión detallada de cada una de las capas y sus interfaces, se va a dedicar un apartado a cada una de ellas.

Capa de aplicación

Constituye la capa que cuenta con un mayor nivel de abstracción. Está compuesta por un conjunto de aplicaciones que utilizan los servicios proporcionados por la capa de control para definir y gestionar el comportamiento de la red subyacente. Por ejemplo, la capa de control cuenta con un servicio de descubrimiento topológico que es utilizado por la capa de aplicación para calcular rutas entre dispositivos de la red. Su programación se basa en lenguajes de alto nivel como Java, Python o C++ con el objetivo de desarrollar aplicaciones multi-plataforma, potenciando la reutilización del software. Sin embargo, en la práctica, cada controlador utiliza un lenguaje de programación y una Application Programming Interface (API) propia que dificultan la reutilización del software.

Capa de control

Esta es la capa que contiene la inteligencia de la red y cuenta con mayor entidad en la arquitectura SDN. Se encarga de orquestar todos los procesos que ocurren en la red, encaminamiento, descubrimiento topológico, supervisión, recuperación ante fallos, etc. Además, actúa de *middleware* entre la capa de aplicación y la capa de infraestructura, traduciendo órdenes de alto nivel en sentencias simples entendibles por los switches, y viceversa. Su estructura permite gobernar muchos dispositivos de la capa de infraestructura, garantizando un crecimiento escalable de la red.

La combinación entre la capa de aplicación y la capa de control da como resultado el controlador de la red. En la actualidad coexisten en torno a veinte plataformas SDN distintas, entre las cuales destaca la hegemonía de Open Network Operating System (ONOS), al ser la plataforma oficial que soporta la ONF y que cuenta con soporte de compañías la talla de Google.

Capa de infraestructura

La capa de infraestructura está compuesta por dispositivos de reenvío de paquetes (switches), los cuales contienen un agente SDN que incluye funciones para comunicarse con el controlador a través de la interfaz southbound. Como ya se ha dejado entrever en párrafos anteriores, esta capa carece de inteligencia, y se limita al reenvío de paquetes atendiendo a las instrucciones impuestas por la capa de control. Entre sus funciones, se encuentra el reenvío o descarte de paquetes, o la recolección de estadísticas sobre el número de paquetes que procesan. Estas funciones tan sencillas reducen los requisitos hardware de los equipos y, por tanto, el coste de los equipos, lo que favorece la economía de escala de las redes SDN.

Interfaz northbound

La interfaz northbound es el canal de comunicación entre la capa de aplicación y la capa de control, y su objetivo es proporcionar una interfaz al administrador de la red a través de la cual pueda gestionar la red SDN abstrayéndole de las operaciones a bajo nivel. Esta interfaz depende del controlador elegido, quien proporciona una API a través de la cual el administrador puede interactuar con la capa de control. Por lo tanto, se trata de una interfaz que no está estandarizada.

Interfaz southbound

A diferencia de la interfaz northbound, la interfaz southbound sí que cuenta con un estándar de facto, OpenFlow [McKeown et al., 2008], y propuestas como P4Runtime [P4 Language Consortium and others, 2017] que apuntan a convertirse en el próximo referente de la interfaz southbound.

OpenFlow es el protocolo de comunicación más utilizado en la interfaz southbound desde el comienzo de la era SDN. Sus orígenes se remontan al año 2008 en la Universidad de Standford, y su historia ha ido evolucionando a lo largo de los años hasta llegar a la versión 1.5.1, lanzada en Abril de 2015. OpenFlow está presente tanto en el controlador como en la capa de infraestructura a través un agente OpenFlow, encargado de comunicar el plano de control con el plano de datos. Los switches OpenFlow cuentan con un proceso denominado *pipeline*, encargado de recorrer un conjunto de tablas para buscar coincidencias con los paquetes que recibe el switch. Los campos de coincidencia en las tablas de los switches los establece el controlador a través del protocolo OpenFlow, indicando el tipo de parámetro a buscar, su prioridad y la tabla en la que queda reflejado, junto con la acción que ha de ejecutar el switch, en el caso de encontrar una coincidencia. Los campos de coincidencia pueden ser direcciones Media Access Control (MAC), direcciones IP, número de identificación del protocolo, puertos, etc., mientras que algunas de las acciones se centran reenvío del paquete hacia otras interfaces del switch o hacia el controlador, y el descarte del paquete. Por ejemplo, el controlador puede indicar que los switches busquen coincidencias de direcciones MAC con un nivel de prioridad 2 en la tabla 2 y que cuando encuentre un paquete coincidente lo descarte. El switch, por cada paquete que recibe, hace una búsqueda por tablas y niveles de prioridad, y si coinciden las direcciones MAC del paquete con los parámetros almacenados en la tabla 2, lo descarta. No obstante, las tablas también disponen de contadores diseñados para recoger valores estadísticos de los paquetes procesados como, por ejemplo, contar el número de paquetes procesados de un determinado flujo.

La comunicación entre el controlador y los switches se realiza a través del canal OpenFlow con tres tipos de mensajes, de controlador a switch, asíncronos y simétricos. Los mensajes de controlador a switch no tienen por qué tener un mensaje de respuesta, y su objetivo es que el controlador configure o consulte reglas o estados de los switches.

Los mensajes asíncronos cubren una amplia variedad de acciones que abarcan todas las notificaciones del estado del switch, se inician en el switch y llevan una respuesta asociada. Por último, los mensajes simétricos son también mensajes que llevan una respuesta asociada sin embargo, se pueden iniciar bien en el controlador, bien el switch, y sirven para comunicar que se han añadido nuevo switches a la red, para indicar que los switches siguen vivos, y para intercambiar información hardware, y software entre el controlador y los switches (características hardware de los switches, versión del protocolo utilizada, estadísticos, etc.). Además, este último grupo incluye un campo reservado para mensajes experimentales que doten de nuevas funcionales a OpenFlow no previstas previamente. La comunicación del canal OpenFlow utiliza el protocolo de transporte TCP para asegurar la entrega de mensajes, y se contempla el uso de TLS para cifrar el canal y encriptar la comunicación entre el controlador y los switches.

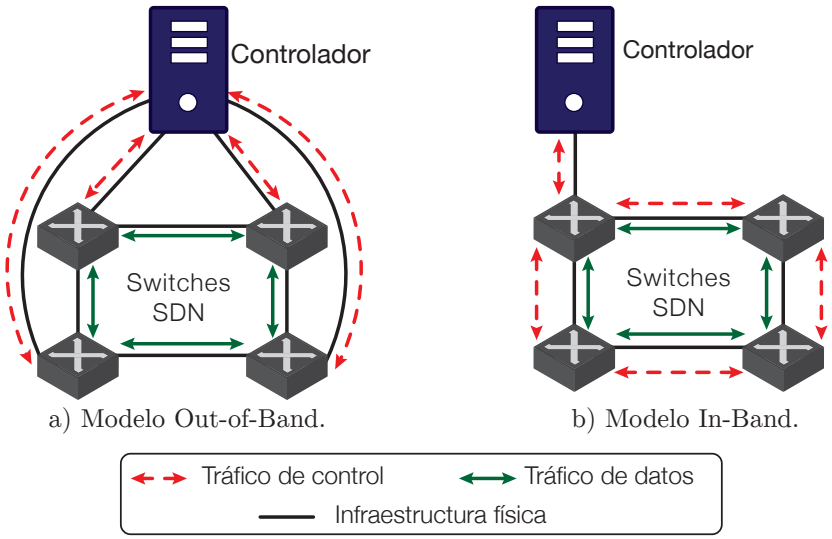
Las limitaciones presentes en OpenFlow [Goransson et al., 2014] han provocado la aparición de soluciones alternativas para la interfaz southbound, entre las que destaca P4Runtime, una propuesta de Google presentada en el Workshop de P4 del año 2007, y cuyo proyecto está actualmente ligado al grupo de trabajo P4.org. En concreto, el lenguaje P4 nace con el objetivo de crear una capa de infraestructura programable a través de lenguaje universal de alto nivel, mediante la definición de acciones de reenvío, procesado, o descarte de paquetes. Sin embargo, P4 se centra solamente en en la programabilidad de la capa de infraestructura, dejando olvidada la comunicación entre el plano de datos y el plano de control. La propuesta encargada de realizar esta comunicación entre las capas de control e infraestructura es, precisamente, P4Runtime, que define los procesos y metodología para configurar, interaccionar o modificar las tablas P4 de la capa de infraestructura desde la capa de control utilizando la tecnología google Remote Procedure Call (gRPC). P4Runtime tiene cinco tipos de operaciones distintas para realizar todos los procesos de comunicación entre el controlador y los switches: Write, Read, SetForwardingPipelineConfig, GetForwardingPipelineConfig, and StreamChannel. Mientras que Write y Read se utilizan para escribir y leer los objetos P4 (tablas, registros, etc.), las operaciones Set/GetForwardingPipeline se utilizan para enviar y recibir los programas P4 de los switches. Estas funciones son nuevas, ya que, hasta ahora, el plano de datos permanecía estático y solamente era posible configurar sus registros, sin embargo, con P4 es posible cambiar la funcionalidad de dicho plano para agregar nuevas funciones o modificar el plano de datos existente. Por último, StreamChannel ofrece un canal de comunicación bidireccional y asíncrono entre la capa de aplicación y la capa de infraestructura para desarrollar funciones similares a las definidas por OpenFlow (envío de mensajes entre controlador y switches, verificación de persistencia del canal, comunicaciones de caducidad y caídas del servicio, etc.). P4Runtime junto con el lenguaje P4 parecen llamados a ser el estándar a seguir en la interfaz southbound debido a la optimización de los procesos gRPC y a la programabilidad adicional que aporta esta solución al plano de datos.

2.1.2 Arquitectura de red física en redes SDN

Tras mostrar la estructura lógica de las redes SDN, se describen sus conexiones físicas, para entender, desde un punto de vista práctico, el esquema real que implica el despliegue de una red SDN, viendo los canales físicos de comunicación que conectan al controlador con la capa de infraestructura, así como la estructura física de la capa de infraestructura.

Como ya se adelantó en el apartado anterior, las redes SDN están formadas por un elemento central, el controlador, y varios equipos de conmutación de paquetes distribuidos por la capa de infraestructura, los switches SDN. No obstante, es posible la coexistencia de varios controladores para dotar de valor añadido a la red SDN como, por ejemplo, labores de respaldo para asegurar la fiabilidad de la red. Por lo tanto, las redes SDN poseen dos tipos de conexiones físicas, conexiones entre los switches de la capa de infraestructura, y conexiones entre el controlador y los switches bajo su dominio. El primer tipo de conexiones forman la topología de red per se, y su morfología depende del ámbito y el propósito al que esté destinada la red SDN. Por ejemplo, es habitual que la capa de infraestructura presente una estructura jerárquica y regular si la red SDN pertenece a un centro de datos, ya que este tipo de red permite ampliar fácilmente su tamaño ante demandas de tráfico. Ejemplos típicos de redes jerárquicas son SpineLeaf [Alizadeh et al., 2014], Vl2 [Greenberg et al., 2009], Fat-Tree [Al-Fares et al., 2008], Portland [Mysore et al., 2009] y Facebook-Altoona [Farrington & Andreyev, 2013]. No obstante, también es posible desplegar redes SDN para cubrir grandes áreas de terreno como, por ejemplo, las redes ISP Pan-Europea [De Maesschalck et al., 2003] o Rogers Fiber BackBone [Rogers Communications Inc., 2018], cuya estructura se aleja de formas regulares y jerárquicas, dando lugar a una estructura irregular que se adapta a la disposición del terreno. No obstante, es el segundo tipo de conexiones (entre el controlador y los switches) la que suscita el interés de esta Tesis, por lo que se profundiza con mayor detalle en ellas.

Existen dos posibles modelos de conexión entre el controlador y los switches, el modelo Out-of-Band, y el modelo In-Band, que se ilustran de forma gráfica en la Figura 2.2, pudiéndose obtener combinaciones híbridas que mezclan ambos modelos. La parte izquierda (Figura 2.2a), recoge la disposición física de las conexiones entre el controlador y los switches para el modelo Out-of-Band, mientras que la mitad derecha (Figura 2.2b) lo hace para el modelo In-Band. En ambas figuras se observa a un controlador que tiene bajo su dominio a cuatro switches, sin embargo, en el modelo Out-of-Band cada switch cuenta con una estructura física dedicada exclusivamente al plano de control, y otra específica para el plano de datos, a diferencia del modelo In-Band, que únicamente cuenta con una estructura física que es compartida por el plano de control y el plano de datos. A priori, el modelo Out-of-Band parece idóneo para este tipo de redes si se toma al pie de la letra la arquitectura lógica de las redes SDN, ya que el plano de control y el plano de datos están completamente aislados entre sí, reservando en exclusiva un canal de comunicación para el plano de control y otro para el plano de datos, lo que



a) Modelo Out-of-Band. b) Modelo In-Band.

Figura 2.2: Modelos de conexión física entre controlador y switches SDN.

garantiza la separación de planos que formula la arquitectura lógica. Sin embargo, en la práctica, el modelo Out-of-Band presenta serias limitaciones de escalabilidad al tener que desplegar una red paralela desde cada switch hasta el controlador para encaminar el tráfico de control, escenario que incrementa los costes de infraestructura y que lo hace inviable en redes que abarcan grandes extensiones de terreno. Por lo tanto, el modelo In-Band se presenta como una alternativa escalable ideal para la mayoría de escenarios, ya que despliega una única red física que es compartida por el plano de control y el de datos. La separación de planos en el modelo In-Band se hace de forma lógica, priorizando el tráfico de control sobre el tráfico de datos para asegurar la operabilidad de la red. Además, el modelo In-Band aumenta la fiabilidad de la red al poder aprovechar la multiplicidad de caminos para encaminar el tráfico de control, pudiendo buscar caminos alternativos en caso de que ocurra un fallo de red. No obstante, a pesar de las ventajas que presenta el modelo de comunicación In-Band, éste conlleva una mayor complejidad de configuración, puesto que es necesario establecer las rutas que comunican al controlador con cada uno de los switches. Además, no existe un estándar oficial para realizar esta configuración automáticamente, por lo que muchos de los despliegues In-Band se basan en ineficientes configuraciones manuales. Por lo tanto, el diseño de un mecanismo automático que establezca las rutas entre el controlador y los switches de la red es un tema de interés a tratar en esta Tesis. A continuación se expone un estudio general de las diferentes propuestas que existen en el estado del arte, mostrando las ventajas y los inconvenientes de los trabajos encontrados.

2.1.3 Propuestas basadas en modelos de comunicación In-Band

A menudo, las propuestas centradas en ofrecer sistemas de comunicación para el plano de control en redes SDN In-Band incluyen soluciones multicamino, con el objetivo de proporcionar rutas alternativas de respaldo que subsanen fallos en la red o problemas de congestión, garantizando en todo momento la operatividad de la red. Sin embargo,

estos mecanismos de recuperación solamente son efectivos si transcurren menos de 50ms [Niven-Jenkins et al., 2009] desde que se detecta el fallo hasta que se restaura la comunicación. Por lo tanto, los sistemas de comunicación SDN In-Band deberían ser autoconfigurables e incluir sistemas de protección o restauración de rutas que recuperen la comunicación con el controlador en menos de 50ms. Atendiendo a estas características se ha estructurado la sección en cuatro apartados de acuerdo con la funcionalidad más relevante de cada trabajo: configuración automática, recuperación ante fallos, esquemas de protección ante fallos y otras propuestas de interés.

Configuración automática

Los protocolos que ofrecen mecanismos de comunicación In-Band autoconfigurables para el plano de control no son frecuentes debido, precisamente, a la infinidad de configuraciones posibles que dependen de la estructura física de la red. No obstante, hay propuestas que hacen una apuesta firme por diseñar propuestas autoconfigurables, aunque en ello inviertan un tiempo apreciable. Uno de los pioneros es ResilientFlow [Omizo et al., 2016], apostando por un sistema de comunicación SDN In-Band autónomo que precisa ayuda externa de Open Shortest Path First (OSPF) [Moy, 1998], un protocolo que distribuye información topológica de la red para obtener las rutas de menor coste. La inclusión de OSPF supone intercambios periódicos de información para actualizar cambios topológicos que incrementan el tráfico de señalización y reducen el ancho de banda disponible para el plano de datos. Además, OSPF es un protocolo lento, ya que por defecto sus temporizadores se fijan en valores superiores al segundo, por lo que el tiempo de configuración de ResilientFlow se alarga en el tiempo. FASIC [Su et al., 2017] proporciona un mecanismo de autoconfiguración que consiste en utilizar el procedimiento incluido en el switch software Open Virtual Switch (OVS) [Linux Foundation, 2020]. Este switch contiene reglas ocultas de alta prioridad que permiten el paso de las peticiones Address Resolution Protocol (ARP) generadas por los switches SDN al arrancar para comunicarse con el controlador. Sin embargo, este mecanismo requiere introducir de forma manual en cada switch las direcciones IP del controlador y del switch utilizadas para crear el canal de comunicación del plano de control. Además, la propuesta incluye un sistema de monitorización continuo del estado de la red con el objetivo de proporcionar un sistema robusto que analice el nivel de carga de los enlaces y evite episodios de pérdida de paquetes de control por congestión de la red. La propuesta también incluye un mecanismo de recuperación ante caídas de enlaces de red que reduce seis veces el tiempo de recuperación con respecto al sistema que proporciona OVS (de 30 a 5s) sin embargo, este tiempo está aún muy lejos de los 50 ms que marca [Niven-Jenkins et al., 2009].

Otras propuestas también ofrecen protocolos de comunicación In-Band con pequeñas pre-configuraciones en los switches (direcciones IP y reglas de reenvío prioritarias), pero incluyen la coexistencia de varios controladores en la red SDN, construyendo dos

tipos de árboles de expansión [Schiff et al., 2015, Schiff et al., 2016]. El primer árbol comunica cada controlador con los dispositivos de la capa de infraestructura, por lo que coexistirán tantos árboles como controladores tiene la topología, mientras que el segundo tipo de árbol se centra en la comunicación entre controladores a través de la capa de infraestructura. Al igual que ocurre con el primer tipo de árbol, también coexisten tantos árboles como controladores tiene la topología, garantizando una comunicación bidireccional entre controladores. De esta forma, cada controlador tiene una ruta que le comunica con todos los dispositivos que están bajo su dominio, y otra ruta que le comunica con el resto de controladores. Renaisance [Canini et al., 2017,Canini et al., 2018] también apuesta por un esquema de comunicación In-Band que comunique tanto switches y controladores, como controladores entre sí, pero bajo dos puntos de vista distintos. El primer enfoque hace una apuesta similar a [Schiff et al., 2015, Schiff et al., 2016], mediante una pre-configuración de los dispositivos de red, construyendo un árbol por controlador. Sin embargo, el segundo enfoque elimina esa etapa de pre-configuración y apuesta por una construcción autónoma, cuya iniciativa parte desde el controlador. Aquí, el controlador descubre secuencialemnte nuevos dispositivos, instalando reglas en cada nuevo dispositivo descubierto para encaminar tanto el tráfico de control como los mensajes de descubrimiento de nuevos dispositivos. El resultado final es un árbol de expansión con origen en el controlador que recorre todos los dispositivos de la red. Basándose en el segundo enfoque, [Bentstuen & Flathagen, 2018] proponen un método en tres pasos: identificación y registro de switches, instalación de reglas en los switches intermedios para encaminar el tráfico de control, y descubrimiento y monitorización de la red.

Existen otros trabajos que proponen también un método doble para establecer la comunicación del plano de control en redes In-Band de forma autónoma o semi-autónoma, como [Sakic et al., 2020]. La primera opción utiliza una aproximación similar a otras propuestas, parte de una pre-configuración de los switches y construye de forma secuencial el árbol de expansión cuyo origen reside en el controlador. En primer lugar, se registra el switch que está directamente conectado con el controlador, creando el primer enlace del árbol y, posteriormente, se incorporan progresivamente el resto de switches en función de la distancia al controlador. De esta forma, se añaden nuevas ramas al árbol de forma secuencial que aprovechan el árbol previamente construido. La segunda opción cuenta con un proceso automatizado, aunque para ello necesita que se haya construido previamente un árbol de expansión desde el controlador con el protocolo RSTP. Tras crear el árbol existe una ruta desde el controlador que llega a todos los dispositivos bajo su dominio, por lo que los switches pueden iniciar simultáneamente la conexión con el controlador. Cuando todos los switches han establecido la comunicación con el controlador, se elimina el árbol generado por el protocolo RSTP para que no interfiera con la red SDN. No obstante, el tiempo invertido para establecer la comunicación entre el controlador y los switches se demora por el uso de temporizadores en RSTP.

ConForm [Freitas et al., 2020] propone un protocolo de comunicación In-Band

autoconfigurable que integra, además, un servicio de descubrimiento topológico, utilizando solamente de cuatro tipos de mensajes: registro de switches en la red SDN, actualización topológica, comprobación del estado y reinicio del servicio. La metodología utilizada para construir el árbol de expansión es similar a trabajos anteriormente mencionados, construyendo el árbol de expansión de forma secuencial desde el controlador, en el que se van registrando los switches en la red en orden incremental de distancia al controlador. No obstante, esta propuesta configura de forma rápida el canal de control, gracias a la ausencia de temporizadores utilizados en otras propuestas para crear el árbol de expansión.

Asadujjaman [Asadujjaman et al., 2018] propone un método automático de conexión por parte de los switches a la red SDN mediante el uso de dos mensajes. La metodología es igual que en trabajos anteriores, se agregan nuevos dispositivos a la red en función de la distancia al controlador (de menor a mayor distancia), para asegurar que los mensajes de los nuevos switches llegan hasta el controlador. Al conectarse un nuevo dispositivo a la red, éste envía un mensaje de descubrimiento a todos sus vecinos, quienes incrementan el coste del mensaje y lo reenvían al controlador. El controlador responde a todos los mensajes, y en el viaje de vuelta, los switches intermedios vuelven a incrementar el coste del mensaje. Finalmente, el nuevo switch recibe todos los mensajes de respuesta y elige la ruta de menor coste como ruta para el canal de control. Sin embargo, la ruta creada es unidireccional (desde switch hasta controlador), por lo que el sentido inverso de la comunicación (desde el controlador hacia los switches) se realiza incluyendo la información de la ruta en el paquete (source routing), lo que incrementa la sobrecarga de datos. Además, esta propuesta incluye un sistema basado en la construcción de anillos que ofrece resistencia ante la caída de múltiples fallos.

Recuperación ante fallos (modo reactivo)

Los sistemas de recuperación ante fallos en redes de comunicaciones detectan fallos en los elementos de la red (enlaces de comunicación o dispositivos de red), y actúan para recuperar la comunicación en el menor tiempo posible, con el objetivo de minimizar los daños provocados por la caída del sistema. Son varios los trabajos que se centran en este problema, y ofrecen soluciones para redes SDN In-Band que restauren de forma rápida eventuales caídas del sistema como, por ejemplo, [Park et al., 2017], quienes establecen un mecanismo de recuperación ante fallos que actúa en función de las condiciones de la red. Tras detectar el fallo, el controlador busca caminos alternativos y puntos de actuación para estimar el retardo de estas alternativas mediante el envío de ráfagas de datos. Tras enviar las ráfagas mide el retardo de las rutas alternativas, y se queda con aquella que ofrece un menor retardo.[Dong et al., 2017] proponen un esquema de recuperación ante fallos en el que el controlador busca un ruta alternativa a la ruta original afectada por el fallo de red que no garantiza una recuperación en menos de 50ms.

RASCAR [Savas et al., 2018] propone un sistema de recuperación ante fallos en

redes con múltiples controladores que resuelve situaciones con fallos parciales de red en elementos clave del sistema (controladores, switches y enlaces de red). Para ello agrupa los dispositivos en islas, minimizando el número de dispositivos que se quedan aislados fuera de las islas. De esta forma, el sistema consigue recuperar regiones enteras restaurando solamente la sección del camino previa al fallo, evitando el cálculo de nuevas rutas para el conjunto de dispositivos conectados a la red.

MORPH [Sakic et al., 2018] propone un framework tolerante a fallos para el plano de control SDN que soporta múltiples controladores, donde tanto la comunicación entre controladores como la comunicación entre controlador y switch se hace In-Band. El framework es capaz de detectar tanto el fallo de los elementos del sistema (controladores y switches), como la presencia de código maligno en los controladores del sistema. Tras la detección del fallo, el sistema reasigna rutas evitando los dispositivos afectados por tal fallo. Tanto la detección de fallos y código maligno, como la reasignación de rutas, dependen de un algoritmo compuesto por varios módulos que configura dinámicamente la red, minimizando la sobrecarga del plano de control.

[Chan et al., 2018] proponen un sistema alternativo de restauración de caminos basado en el diálogo entre controladores ubicados en el mismo dominio de red. Para ello, el algoritmo designa un controlador maestro y uno esclavo, y calcula las rutas que comunican cada controlador con cada uno de los switches de la red intentando, en la medida de lo posible, que los conjuntos de caminos entre los controladores y un mismo switch sean disjuntos para mejorar el tiempo de recuperación. Su mecanismo de detección de fallos es capaz de ubicar el elemento que presenta el fallo, así como clasificar su tipo (switch o enlace de comunicación). La ubicación de los elementos de fallo se consigue con el envío periódico de mensajes que recorren toda la red y tienen como origen y destino el controlador maestro, que detecta el fallo si pasado un tiempo no recibe respuesta, mientras que el descubrimiento del tipo de dispositivo se realiza enviando mensajes de control por parte de ambos controladores (maestro y esclavo) a todos los switches, si no contesta alguno de ellos el elemento de fallo es un switch, en caso contrario es un enlace. Una vez que se ha ubicado el fallo, el controlador maestro procede a restaurar la ruta en los elementos que presentan el fallo en el caso de que pueda comunicarse con ellos, en caso contrario, los controladores maestro y esclavo dialogan para que el controlador esclavo instale las nuevas rutas. Este último caso denota la importancia de que los caminos entre distintos controladores y el mismo switch sean disjuntos, ya que así se garantiza la comunicación de alguno de los controladores con el switch, lo que evita perder la comunicación del plano de control.

Esquemas de protección de caminos

Los esquemas de protección de caminos apuestan por reservar recursos a priori (un camino principal más n caminos de respaldo), para después subsanar caídas del servicio con una simple conmutación de rutas. Este enfoque reduce el tiempo de recuperación ante fallos del sistema con respecto a los enfoques reactivos, ya que aquí, basta con conmutar de la ruta principal a la ruta de respaldo para restablecer la comunicación, mientras que los sistemas de restauración tienen que buscar la ruta y hacer la posterior conmutación. Sin embargo, los esquemas de protección desaprovechan recursos de la red debido a su reserva previa y no conocen el estado de la red en el momento del fallo, por lo que la ruta de respaldo puede estar desfasada o afectada por otros posibles fallos. [Dong et al., 2017], además de proponer un sistema de recuperación reactivo, propone un segundo sistema de protección activo que utiliza el concepto de tabla de grupo de OpenFlow para establecer a priori un camino principal y otro de respaldo, consiguiendo reducir el tiempo de recuperación por debajo de los 50ms. En caso de fallo, el sistema conmuta al camino de respaldo sin que el controlador tenga que añadir nuevas rutas en los switches, ya que éstos tienen las reglas preasignadas.

[Hu et al., 2014] propone un sistema de protección mediante camino de respaldo que maximiza los caminos de control existentes. En caso de fallo, el dispositivo conmuta inmediatamente a la ruta de respaldo, traspasando el tráfico de control a un vecino que tiene un canal de comunicación activo con el controlador. Las técnicas de encaminamiento por las que apuesta son *Local Rerouting* (reenvío a un switch con un canal de comunicación bidireccional activo hacia el controlador) y *Reverse Path Forwarding* (reenvío a un switch con un canal de comunicación descendente desde el controlador que a su vez tenga un vecino con un canal de comunicación activo con el controlador), siendo prioritaria la primera debido a que requiere un menor número de saltos para llegar al controlador. [Huang et al., 2016] propone un sistema de respaldo que tiene en cuenta los costes de la red para minimizar el uso de recursos y evitar los problemas de congestión que acusan las propuestas basadas en *Local Rerouting* y *Reverse Path Forwarding* ante fallos en la red. Sin embargo, a pesar de sus esfuerzos en optimización de recursos, el sistema solo recupera las sesiones de control ante caídas en un único enlace de la red.

[Goltsmann et al., 2017] apuestan por un esquema de protección distribuido basado en el cálculo de caminos de respaldo al camino principal. Para ello, primero obtiene la ruta principal del canal de control (árbol de expansión bidireccional con origen en el controlador), y posteriormente, obtiene dos rutas de respaldo desde cada uno de los switches de la red hasta su padre y su abuelo en el árbol de expansión, respectivamente. De esta forma, se obtienen caminos alternativos para subsanar fallos en el enlace previo del árbol (camino de respaldo hasta nodo padre) y fallos en el nodo padre (camino de respaldo en el nodo abuelo). La conmutación de ruta en caso de fallo es inmediata, y siempre requiere actuaciones desde el nodo hijo posterior al fallo para reestablecer la

comunicación con el árbol de expansión, y por ende, con el plano de control. Sin embargo, la propuesta solo plantea la idea base del protocolo de comunicación In-Band, describiendo todo el proceso con mayor nivel de detalle en [Holzmann & Zitterbart, 2019].

Otros trabajos apuestan firmemente por mantener en paralelo varios canales de comunicación activos, garantizando una comunicación continua entre las capas de control e infraestructura. Por ejemplo, [González et al., 2018] utiliza simultáneamente varios canales de comunicación entre el controlador y los switches basándose en el hecho de que no todos los canales de comunicación se verán afectados ante un fallo de la red, lo que asegura una comunicación permanente entre controlador y switches. Además, refuerza dicha teoría al utilizar rutas disjuntas, aislando los caminos entre sí. La búsqueda y el establecimiento de los múltiples caminos comienza tras la unión satisfactoria de los switches a la red (el controlador puede comunicarse con todos los switches que hay bajo su dominio). Después de encontrar múltiples caminos, el algoritmo selecciona aquellos que considera óptimos intentando garantizar disjuntividad entre ellos. Más adelante, el algoritmo divide el flujo que transporta el tráfico de control en varios subflujos, y asigna un camino a cada subflujo con ayuda del protocolo MultiPath TCP (MPTCP) [Ford et al., 2013]. El uso de MPTCP junto con la característica de disjuntividad de caminos es un recurso que también utiliza [Raza & Lee, 2019], sin embargo, esta propuesta solamente se centra en el algoritmo de búsqueda de los caminos disjuntos, dejando para trabajos futuros la definición del protocolo In-Band.

La disjuntividad entre caminos como característica de respaldo es también un recurso utilizado en los esquemas de protección, ya que se garantiza que el camino de respaldo esté aislado del camino principal. [Liao & Tsai, 2018] propone un algoritmo que busca múltiples caminos disjuntos en enlaces si la topología tiene suficiente conectividad. Tras obtener el grafo topológico, el controlador calcula los caminos total o parcialmente disjuntos entre controlador y switches, e instala las reglas necesarias del camino de respaldo para encaminar el tráfico de control, en caso de fallo, sin la intervención del controlador. [Mohan et al., 2018] van más allá proponiendo un protocolo de comunicación In-Band que detecta nodos maliciosos en la red a través del análisis de sus mensajes y, ofrece, además, un esquema de protección con un camino de respaldo disjunto en nodos desde cada switch al controlador. Sin embargo, la carga computacional que genera el algoritmo hace que solo sea viable en redes de pequeña escala.

Otras propuestas de interés

Este apartado muestra propuestas que están relacionadas con la temática In-Band pero que no encajan en las categorías anteriores. El tráfico de control en redes In-Band requiere de la instalación de reglas en los switches para comunicar las capas de control e infraestructura, ya que el tráfico de control es encaminado a través de la infraestructura física de la red. En las redes SDN clásicas el controlador establece un canal de

comunicación único con cada switch que hay bajo su dominio, mientras que en las redes In-Band esta comunicación utiliza los recursos físicos de la capa de infraestructura.Por lo tanto, en el modelo In-Band, muchos de los switches actúan como switches intermedios encargados de reencaminar el tráfico de control de otros switches. Como se ha visto durante este capítulo, la mayoría de las propuestas utilizan árboles de expansión para encaminar el tráfico de control y evitar las temidas tormentas de mensajes. El efecto indeseado que provoca el uso de árboles de expansión es que los switches más próximos a la raíz del árbol aglutinan más tráfico de control proveniente de otros switches de la red, por lo que el número de entradas que necesitan dichos switches para encaminar todo el tráfico de control se incrementa. En redes de gran tamaño esto se puede descontrolar, ya que cada flujo de control necesita dos entradas por cada switch atravesado (una en sentido ascendente y otra en sentido descendente). Gather [Yan et al., 2017] aborda este problema planteando un algoritmo que reduce el número de reglas necesarias para encaminar el tráfico de control, basándose en la unificación de reglas y agrupación de flujos, mediante el cual una única regla puede encaminar el tráfico de control de varios switches. El algoritmo reduce en torno a un 40 % el número de reglas necesarias, mientras que solamente aumenta un 5 % el tiempo de procesamiento, garantizando una optimización de recursos en la red. [Hark et al., 2017] también proponen un sistema que reduce el número de reglas para encaminar tráfico de control In-Band, pero enfocado a la comunicación entre controladores, utilizando redes de área local virtuales, del inglés Virtual Local Area Network (VLAN), y una codificación especial en la identificación de los controladores que denominan *One-Hot-Coding*.

Otro enfoque interesante en redes In-Band es el encaminamiento activo del tráfico de control, cuyo objetivo consiste en minimizar las pérdidas de comunicación en el canal, o evitar incrementos en la latencia del tráfico de control por congestión de enlaces, ya que ambos planos (control y datos) comparten los mismos recursos físicos. [Fan & Yang, 2020] apuestan por la aplicación de técnicas de Quality of Service (QoS) junto con una monitorización activa de la red para reencaminar el tráfico de control por los enlaces de la red menos cargados. Además, no elimina las rutas por defecto creadas al principio de la comunicación In-Band para que el sistema pueda actuar en caso de caída, aunque este aspecto no presenta mayor relevancia debido a que la propuesta precisamente evita caídas de los enlaces por congestión. Sin embargo, la escalabilidad es un factor limitante ya que se monitoriza y se calculan las rutas óptimas para el tráfico de control continuamente, lo que implica un análisis computacional apreciable en redes de media y gran escala.

2.1.4 Modelos híbridos de redes SDN

La migración de redes tradicionales hacia soluciones SDN requiere actuaciones parciales en la red debido a los costes de inversión y al cambio de paradigma que supone esta transformación, pasando de un modelo completamente descentralizado a un modelo totalmente centralizado. Es por ello que la migración hacia SDN se hace progresivamente, introduciendo en las redes heredadas nuevos dispositivos SDN que conviven hasta el desmantelamiento completo de la red tradicional, conociéndose este concepto como red híbrida (hibridación en horizontal). Algunos fabricantes ya han diseñado propuestas OpenFlow retrocompatibles con dispositivos tradicionales, [Brocade, 2015,Brocade, 2012], mientras que otros investigadores han estudiado el proceso de migración hacia modelos SDN, centrándose en los costes de migración [Poularakis et al., 2017], o analizando características de ingeniería de tráfico como son el balanceo de carga y el tiempo de recuperación ante fallos [Hong et al., 2016]. Pero no solo las redes híbridas están formadas por la convivencia de dispositivos SDN y no SDN, sino que también son redes híbridas aquellas que aún estando compuestas en su totalidad por dispositivos SDN, realizan ciertas funciones de forma distribuida sin la intervención del controlador (hibridación en vertical). Precisamente, es este último modelo "híbrido vertical" el que despierta mayor interés en la presente Tesis Doctoral, debido a las ventajas que ofrece frente a un enfoque SDN puro:

- Reduce el tráfico de control al suprimir la participación activa por parte del controlador. No es necesario que el controlador intervenga de forma activa en el proceso, solo tiene que interactuar con la capa de infraestructura para disparar el proceso y recoger los resultados.
- Reduce la sobrecarga de cómputo en el controlador. El cómputo que realizaba un único dispositivo se distribuye por la capa de infraestructura. El controlador ahora solo se encarga de recoger y procesar la información recopilada.
- Incrementa la fiabilidad del proceso al distribuirlo entre los dispositivos de la capa de infraestructura. Se evitan errores provocados por fallos de equipamiento ya que la operación no se interrumpe si un equipo falla, al contrario que en un enfoque SDN puro, donde un fallo del controlador anula el proceso. Además, al liberar al controlador de carga computacional, éste puede destinar la potencia de cálculo sobrante a la monitorización exhaustiva del estado de la red, pudiendo anticiparse a problemas que colapsen la red.

TEDP [Rojas et al., 2017] propone un sistema de descubrimiento topológico para redes SDN en el que el controlador solamente obtiene la topología recogiendo la información que le brindan los switches de la capa de infraestructura. Estos switches tienen la autonomía suficiente para iniciar el proceso de descubrimiento y explorar toda la red a través de la difusión de un mensaje de exploración. Además, el proceso de exploración está diseñado para no generar bucles que desencadenen tormentas de mensajes que saturen la red. Junto

con la topología, el protocolo aporta como valor añadido el árbol de mínima latencia que recorre toda la red. eTDP [Ochoa-Aday et al., 2019] realiza un proceso similar, distribuyendo las funciones de descubrimiento topológico entre los dispositivos de red mediante un agente eTDP.

Otra de las aplicaciones en las que las redes híbridas tienen cabida, es en la recuperación de caminos ante fallos de red, minimizando los tiempos de recuperación, ya que se evita el tiempo de notificación de fallo al controlador, la posterior búsqueda de ruta y la asignación del nuevo camino. [Alvarez-Horcajo et al., 2017] diseñan un sistema de recuperación ante fallos de red que trabaja de forma distribuida, sin asistencia del controlador o con una asistencia parcial. En el modelo sin asistencia, la capa de infraestructura, tras detectar el fallo, realiza una exploración de la red mediante una difusión controlada de mensajes desde el puente afectado por el fallo, encontrando un camino alternativo. En el modelo asistido, el switch que detecta el fallo de red indica al controlador el punto de fallo, para que el controlador indique a su vez a la capa de infraestructura el lugar desde el que hay que restaurar. Tras recibir la orden, la capa de infraestructura realiza un proceso de exploración similar al del primer modelo. Ambos modelos mejoran el tiempo de recuperación frente a una solución basada en controlador, siendo el modelo completamente híbrido (sin asistencia del controlador) el que restaura el fallo en menor tiempo. [Tilmans & Vissicchio, 2014] utilizan el protocolo distribuido Interior Gateway Protocol (IGP) [Rekhter, 1988] para restaurar rápidamente fallos de red. Los switches están configurados para intercambiar información de enrutamiento a través del protocolo IGP y establecer rutas de respaldo con dicha información, para, en caso de fallo, conmutar directamente a la ruta de protección evitando así la intervención del controlador.

Por último, otros enfoques apuestan por mejorar la escalabilidad de la red, como es el caso de [Xu et al., 2018], donde estudian distintos modelos de redes híbridas para terminar diseñando un modelo de encaminamiento que integra el encaminaminento tradicional y el enfoque SDN, obteniendo una red escalable con un rendimiento óptimo que reduce el número de entradas de reenvío del plano de datos.

2.2 Múltiples caminos en redes de comunicaciones

Uno de los problemas clásicos de las redes de comunicaciones se centra en la búsqueda de rutas con menor número de saltos o menor coste, debido a los beneficios que aportan [Krishnan & Silvester, 1993, Cidon et al., 1999]. Este tipo de estudios siguen vigentes en la actualidad, sin embargo, la búsqueda se ha adaptado a las redes de nueva generación y se ha añadido el problema del multicamino como valor añadido. Las soluciones multicamino aportan claros beneficios al incluir múltiples soluciones monocamino. Estas rutas adicionales ofrecen cualidades adicionales que mejoran el comportamiento de la red en general, bien por distribuir mejor la carga de tráfico a través

de varios caminos, bien por incrementar la fiabilidad al proporcionar rutas de respaldo, o por incrementar la seguridad de los datos transportados al poder dividir un flujo de datos entre varios caminos, entre otras opciones. Además, todos estos beneficios se incrementan si las rutas no comparten elementos entre sí, o lo que es lo mismo, las rutas son disjuntas.

En cuanto al proceso de búsqueda, éste se puede realizar de forma distribuida o de forma centralizada. Mientras que en un entorno distribuido existe una participación activa entre todos los dispositivos de la red en la que se intercambia información entre vecinos y se procesan las rutas a nivel local, en un entorno centralizado el proceso de búsqueda lo realiza solamente un dispositivo, encargado de calcular las rutas basándose en la información topológica recogida previamente por algún protocolo o algoritmo externo. Ambos modelos cuentan con cualidades positivas y negativas que los hacen aptos para unos entornos específicos, ya que no siempre es posible aplicar uno u otro enfoque. Para conocer con mayor detalle cada enfoque, a continuación se detallan sus ventajas y sus inconvenientes.

- Ventajas de los sistemas distribuidos:
  - Aumento de la fiablidad. El modelo de cómputo distribuido lo hace proclive a minimizar fallos endémicos que anulen el sistema completo, ya que si un elemento del sistema falla, los demás siguen funcionando.
  - Sistemas escalables. A pesar de la complejidad añadida que supone la adaptación de algoritmos de búsqueda a entornos distribuidos, su escalabilidad (generalmente) está garantizada, precisamente, por la distribución de la carga computacional entre todos los dispositivos de la red. Además, un incremento en el número de dispositivos disminuye la carga computacional global puesto que se incrementa el número de elementos entre los que distribuir el trabajo.
- Inconvenientes de los sistemas distribuidos:
  - Incremento del tiempo de convergencia. Si el proceso de distribución no es óptimo y las características hardware de los equipos están limitadas, se puede incrementar el tiempo invertido por el algoritmo/protocolo para obtener las soluciones. Este hecho hace que a veces sea contraproducente adoptar soluciones distribuidas para la búsqueda de rutas.
  - Visión incompleta del problema. Al procesar información localmente no se tiene una visión completa del problema que facilite el proceso de búsqueda.
- Ventajas de los sistemas centralizados:
  - Simplicidad en el despliegue, manejo y control. Los algoritmos centralizados suelen seguir un proceso monolítico secuencial que acelera los tiempos de desarrollo. Además la depuración y corrección de errores es sencilla, puesto que se actúa en un único dispositivo.

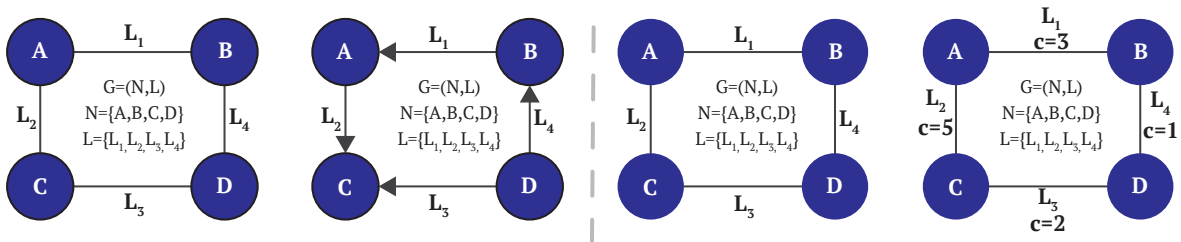
- Inconvenientes de los sistemas centralizados:
  - Problemas de escalabilidad al estar centrado todo el cómputo en un único punto. Incrementos del tamaño de red implican un mayor procesado de información por cada dispositivo, lo que unido a un crecimiento de la complejidad (típicamente) exponencial o logarítmico, hace que el procesado de información en redes grande sea un proceso lento.
  - Sistema vulnerable al existir un punto único de fallo.

Para analizar con mayor detalle el problema del multicamino en redes de comunicaciones, esta sección introduce un apartado con los conceptos básicos de teoría de grafos y la evolución natural que ha sufrido el proceso de búsqueda de múltiples caminos a lo largo de la historia. Después, la sección analiza diferentes propuestas multicamino clasificadas en dos apartados, múltiples caminos, y múltiples caminos disjuntos. Además, esta clasificación se subdivide en otros dos epígrafes en función del procedimiento de búsqueda, enfoque centralizado o distribuido.

2.2.1 Terminología básica

La teoría de grafos es en una disciplina matemática centrada en el estudio de las propiedades de los grafos. Un grafo es una representación visual de datos compuesta por una serie de objetos llamados nodos, conectados a través de otro tipo de objetos denominados enlaces, cuyo origen data del siglo XVIII con el problema de Könisgberg [Euler, 1741]. Formalmente, un grafo se define como $G = (N, L)$, dónde N se corresponde con el conjunto de nodos, y E con el conjunto de enlaces que relacionan el conjunto de nodos. Atendiendo a la direccionalidad del conjunto de enlaces, los grafos se pueden clasificar en grafos dirigidos o no dirigidos, siendo los grafos dirigidos los que restringen el tránsito por los enlaces a una o varias direcciones específicas, mientras que los grafos no dirigidos no establecen ninguna obligatoriedad en la dirección de tránsito. Asimismo, en función del coste que supone atravesar un enlace, los grafos también pueden clasificarse en grafos pesados/ponderados (los enlaces tienen un coste asociado) y no pesados/ponderados (los enlaces no cuentan con un peso). Por último, es posible mezclar ambas clasificaciones en función de la direccionalidad y el peso de los enlaces, dando lugar a grafos no dirigidos y pesados, grafos dirigidos y no pesados, etc. A modo de resumen, la Figura 2.3 muestra visualmente los tipos de grafos expuestos.

Una vez definido tanto el concepto de grafo como su clasificación, se estudian los algoritmos que permiten recorrer dichos grafos o buscar elementos dentro de él, siendo Breadth First Search (BFS) y Deep First Search (DFS) [Cormen et al., 2001] los pioneros que, posteriormente, han servido de inspiración a otros algoritmos. BFS es un algoritmo que realiza una búsqueda ciega recorriendo un grafo no dirigido y no ponderado en "sentido horizontal" desde un nodo raíz (búsqueda en anchura). El algoritmo analiza todos



a) Grafo no dirigido. b) Grafo dirigido. c) Grafo no pesado. d) Grafo pesado.

Figura 2.3: Clasificación de grafos en función de la direccionalidad y el peso de los enlaces.

los vecinos del nodo raíz, y tras examinarlos, explora los nodos del siguiente nivel (sus vecinos). El proceso continua descendiendo en la jerarquía hasta que se han recorrido todos los nodos del grafo. El resultado de esta búsqueda es un árbol enraizado en el nodo origen que recorre todos los nodos del grafo y proporciona el camino con menor número de saltos a cada nodo. En relación a la cantidad de recursos temporales y/o físicos que conlleva esta búsqueda, término conocido como complejidad computacional, es proporcional a la suma del conjunto de nodos y enlaces que componen el grafo ($O(|N|+|L|)$), debido a que, en el peor de los casos, BFS tiene que recorrer todos los nodos y enlaces de dicho grafo. Por su parte, DFS también realiza un proceso de búsqueda en grafos no dirigidos y no pesados partiendo de un nodo raíz, pero en "sentido vertical" (búsqueda en profundidad). Al contrario que la propuesta previa, DFS disminuye de nivel en cada salto, analizando una rama completa del árbol. Cuando se han visitado los nodos de una rama asciende en la jerarquía para explorar otra rama de un nodo previamente procesado. La complejidad computacional de DFS es la misma que la obtenida por BFS, puesto que ambos algoritmos, en el peor de los casos, tienen que analizar todos los nodos y enlaces del grafo. Sin embargo, el árbol obtenido por DFS, al igual que BFS, recorre todos los nodos de la red, pero por contra, no obtiene el camino con menor número de saltos a cada uno de ellos como sí hace BFS.

El algoritmo de Dijkstra [Dijkstra, 1959] es un propuesta enfocada a grafos no dirigidos y ponderados inspirada en el proceso de búsqueda de BFS. Este algoritmo, en lugar de analizar el grafo por niveles de vecindad (analizo el nodo raíz, luego sus vecinos, posteriormente los vecinos de los vecinos, etc.) analiza el grafo en relación al coste acumulado desde el nodo raíz. El proceso de búsqueda comienza en el nodo raíz y va examinando nuevos nodos en relación a su coste acumulado (de menor a mayor coste), actualiza el coste acumulado de nodos vecinos si se mejora el coste anteriormente guardado, e impide que nodos examinados se vuelvan a evaluar. Como máximo, el algoritmo realiza $N-1$ iteraciones, cuyo resultado es el árbol de expansión de coste mínimo con origen en el nodo raíz que recorre todos los nodos del grafo. La complejidad computacional de Dijkstra es $O(N^2)$ [Cormen et al., 2001], sin embargo existen optimizaciones en la gestión de la lista de nodos que mejoran su rendimiento, destacando la utilización de cola de prioridad ($O((|N|+|L|)\cdot log|N|)$ [Cormen et al., 2001], y el montículo de Fibonacci [Fredman & Tarjan, 1987] ($O(|N|\cdot log(|N|+|L|))$ [Cormen

et al., 2001]. Bellman-Ford [Cormen et al., 2001] resuelve el mismo problema con una estructura muy similar al algoritmo de Dijkstra, pero opta por grafos dirigidos y ponderados en los que pueden incluirse además, pesos negativos. Sin embargo, el algoritmo es menos óptimo que el de Dijkstra, por lo que se invierte más tiempo para resolver el problema y la complejidad computacional aumenta hasta $O(N \cdot L)$.

Todos los algoritmos analizados en este apartado obtienen un árbol que recorre todos los nodos del grafo desde un nodo raíz/origen, proporcionando múltiples rutas entre el nodo origen y el resto de nodos del grafo. Sin embargo, este conjunto de soluciones solo incluye una única ruta entre el nodo origen y cada uno de los nodos que componen dicho grafo, por lo que si se quieren obtener múltiples rutas entre un par de nodos es necesario ejecutar varias veces el algoritmo. No obstante, no basta con una simple repetición, sino que es necesario modificar su estructura original para asegurar que las sucesivas ejecuciones proporcionan multiplicidad de caminos entre un par de nodos. Una vez que se ha mostrado la terminología básica, la base sobre la que se cimientan los protocolos de encaminamiento y routing en las redes de comunicaciones, y un método arcaico para la construcción de múltiples caminos entre pares de nodos, se procede a explicar con mayor detalle los protocolos o algoritmos dedicados exclusivamente a la búsqueda de múltiples caminos en redes de comunicaciones.

2.2.2 Propuestas multicamino

El estudio mostrado en este apartado se centra en las propuestas que obtienen múltiples caminos entre un par de nodos/switches. El estudio excluye a las propuestas multicamino que obtienen caminos disjuntos entre sí, puesto que se analizarán en una sección posterior. Además, como se adelantó en el último párrafo de la Sección 2.2, el apartado divide los trabajos en función del escenario en el que se obtienen según correspondan a entornos distribuidos o a entornos centralizados. Mientras que el primer caso está enfocado a redes tradicionales, el segundo focaliza la búsqueda de múltiples caminos en redes SDN, donde el proceso se realiza de forma centralizada en el controlador.

Entornos distribuidos

La búsqueda en entornos distribuidos se realiza mediante procesos totalmente independientes que se encargan de difundir la información topológica (habitualmente por un intercambio activo de mensajes), de procesar las múltiples rutas, y de compartir las rutas obtenidas de forma distribuida en caso de que sea necesario. Todos estos procesos requieren una mayor complejidad, ya que en lugar de centrarse solamente la búsqueda de rutas, también se tienen que diseñar e implementar los mecanismos que distribuyen la información por la red, y trazar la distribución del proceso de cómputo. Por lo tanto, la concepción de protocolos de búsqueda en entornos distribuidos requiere una mayor complejidad que para entornos centralizados.

[Minkenberg & Gusat, 2012] patentan un proceso basado en la difusión de paquetes a través de la red para encontrar las rutas que conectan un switch origen con un switch destino. La propuesta utiliza un mecanismo de exploración basado en la inundación de mensajes que evita tormentas de paquetes provocadas por la aparición de bucles. Dicho mecanismo difunde solamente el primer paquete que recibe un switch desde un switch origen desconocido a través de todas sus interfaces, exceptuando aquella interfaz por la que recibió dicho paquete, además, marca dicha interfaz como prioritaria y la asocia con el switch origen. A partir de ese momento, el switch solo difunde los mensajes del switch origen que entren por la interfaz prioritaria, descartando el resto de paquetes para evitar bucles. Este método hace un aprendizaje en "sentido inverso", ya que en realidad se está aprendiendo el camino hacia el switch origen. El mismo proceso se repite desde el destino, creando un camino en dos sentidos que no tiene por qué compartir el mismo trayecto. Además, la propuesta incorpora un mecanismo de distribución y actualización del coste de los switches por la red mediante la inclusión del coste que tiene el switch atravesado en el paquete difundido. Las tablas de encaminamiento, propias de cada switch, incluyen además una estructura bidimensional en la que se pueden asociar varios puertos del switch a un mismo switch origen/destino para realizar encaminamiento multicamino.

Al igual que Minkerberg y Gusat, [Marukawa et al., 2011] proponen un método de descubrimiento multicamino escalable basado también en técnicas de exploración. Como novedad, incluyen una extensión de la trama estándar de Ethernet, añadiendo un nuevo campo, el tiempo de vida de la trama. Este nuevo campo limita el proceso de exploración a un número máximo de saltos, lo que restringe en mayor medida la difusión de mensajes. El proceso bidireccional incluye una petición de descubrimiento desde el switch origen al switch destino a través del camino de menor longitud, obtenido por otro protocolo. Después el switch destino inicia un proceso de exploración en sentido contrario (de destino a origen), en el que cada switch atravesado incluye el puerto de entrada y salida del mensaje de exploración. Cuando el mensaje llega al switch origen contiene la información completa de la ruta atravesada, por lo que el switch origen solo tiene que leer el contenido del paquete para conocer la información de encaminamiento.

El IEEE bajo el estándar IEEE 802.11aq desarrolla SPB [IEEE, 2012], un protocolo de encaminamiento múltiple en redes de switches que reemplaza a viejos estándares como STP [IEEE, 1998] o RSTP [IEEE, 2001], quienes apostaban por esquemas monocamino. Mientras que STP y RSTP obtienen un árbol de expansión enraizado en un único switch raíz, SPB obtiene múltiples árboles de expansión enraizados en diferentes puntos estratégicos de la red (switch de borde o frontera), para obtener múltiples caminos de mínimo coste desde varios puntos, proporcionando así multiplicidad de caminos. La solución supone un cambio de filosofía radical en las redes de switches, donde, históricamente, para comunicar toda la red, se utilizaban árboles de expansión enraizados en un dispositivo (designado o elegido por la red en función del identificador de switch menor), que infrautilizaba los recursos de la red para evitar la formación de bucles. Esta

nueva visión amplía el horizonte del multicamino en redes de switches y sirve como idea base sobre la que construir nuevas propuestas, como, por ejemplo [Geng et al., 2018], dónde también se construyen varios árboles de expansión de mínimo coste, pero en lugar de crearlos desde switches frontera, se construye un árbol desde cada uno de los nodos que forman la red, aumentando más aún si cabe la multiplicidad de caminos. Esta propuesta cuenta además con dos variantes distintas: un modelo estático que realiza la construcción de los árboles de la red completa, y un modelo dinámico que actualiza los costes asumiendo que el modelo estático ha construido previamente todos los árboles de la red. A pesar de la versatilidad que ofrece y la diversidad de caminos aportados, la sobrecarga computacional que aporta en conjunto no es elevada.

Entornos centralizados

Los algoritmos destinados a la búsqueda de múltiples caminos en entornos centralizados dependen de protocolos o algoritmos externos para obtener el grafo topológico sobre el que trabajan, como por ejemplo [Alsaeedi et al., 2019,Pakzad et al., 2016,Azzouni et al., 2017], que proporcionan protocolos de descubrimiento topológico en redes SDN. No obstante, los protocolos de descubrimiento topológico se escapan del objetivo de la Tesis, por lo que este apartado se centra únicamente en el proceso de búsqueda de los caminos, posterior al descubrimiento toplógico.

Es frecuente encontrar en la literatura trabajos que apuestan por la combinación de técnicas QoS con múltiples caminos, dados los beneficios que aporta a la ingeniería de tráfico el uso de QoS [Srivastava et al., 2004]. DAMR [Jiawei et al., 2018] cuenta con un esquema de encaminamiento múltiple que aplica políticas de QoS para calcular el coste de los enlaces y actualizar el grafo de la red. El controlador SDN realiza una monitorización activa de la red midiendo el retardo de los enlaces, el ancho de banda disponible, las desviaciones temporales en la entrega de los paquetes (*jitter*), y las pérdidas de paquetes, para obtener el camino óptimo entre dos nodos. Este servicio se ejecuta bajo demanda, y su solución depende del estado de la red, por lo que dos demandas de servicio entre los mismos nodos obtienen caminos distintos, de ahí el multicamino. No obstante, esta propuesta se centra más en la descripción de la arquitectura y su integración en SDN que en detallar el algoritmo de optimización para obtener las múltiples rutas. Ahora bien, otras propuestas si que hacen hincapié en dicho proceso y, además, apuestan por la integración de algoritmos clásicos como Dijkstra o BFS con las técnicas de QoS. Uno de esos trabajos es el propuesto por [Smith & Thurlow, 2013], quienes conciben DMSR basándose en el algoritmo de Dijkstra, adaptándolo para trabajar en grafos no dirigidos y ponderados con unos costes de enlace que dependen de las variables anteriormente citadas (retardo, ancho de banda y *jitter*). DMSR obtiene y mantiene actualizado un conjunto de múltiples caminos por los que se encamina el tráfico de la red, cuyo resultado depende de las condiciones del entorno. HiQoS [Yan et al., 2015] también adapta el algoritmo

de Dijkstra a las técnicas QoS que demandan los nuevos servicios de red. Sin embargo, esta propuesta, además de obtener un conjunto óptimo de caminos entre dos puntos, encamina el tráfico por ellos simultáneamente, obteniendo mejoras significativas en el ancho de banda transmitido. De la misma forma, los algoritmos BFS y DFS también se combinan con políticas QoS, como [Hossen et al., 2019], donde se evalúa cuál de los dos algoritmos obtiene mejores resultados al integrarlos con técnicas QoS sobre una red SDN. El resultado de la comparativa da vencedor a DFS en términos de retardo, ancho de banda y Round-Trip Time (RTT). [Ramdhani et al., 2016] también apuestan por el algoritmo DFS para encontrar todas las rutas en una red SDN a las que les aplica una segunda función de optimización para elegir la ruta óptima en términos de ancho de banda. Al igual que los trabajos anteriores, la multiplicidad de caminos entre el mismo par de nodos se obtiene debido a las condiciones cambiantes de la red.

También existen otras propuestas SDN que no se basan en técnicas QoS para obtener múltiples caminos, sino que aplican otra metodología. BFSF [Naseri et al., 2019] limita su ámbito de uso a redes jerarquizadas de centros de datos utilizando un método de búsqueda bidireccional. El proceso realiza, desde los nodos origen y destino, una búsqueda en paralelo del camino de menor coste hacia el nivel superior de la jerarquía, cuya combinación da como resultado el camino final. Al cambiar las condiciones de red y volver a aplicar el mismo algoritmo, los caminos obtenidos serán distintos.

2.2.3 Propuestas multicamino con disjuntividad

La metodología que sigue este apartado es igual que la del apartado anterior, solo que en éste se centra el estudio en una categoría especial de multicamino: los caminos disjuntos entre sí. A modo de recordatorio, existen dos tipos de caminos disjuntos en función de su composición, caminos disjuntos en enlaces y caminos disjuntos en nodos. Mientras que en el primer grupo se impide la reutilización de enlaces por más de un camino, en el segundo se impide la reutilización de nodos y enlaces, a excepción de los nodos extremos. Una vez recordadas las particularidades de este tipo de caminos, se procede a analizar en detalle las propuestas de mayor interés encontradas en la literatura.

Entornos distribuidos

Unos de los pioneros en obtener múltiples caminos disjuntos en entornos distribuidos fueron [Itai & Rodeh, 1988], proporcionando un algoritmo que obtiene k árboles enraizados en un nodo origen que son disjuntos en enlaces, lo que da lugar a que desde cada nodo existan k caminos disjuntos hasta el nodo raíz. Sin embargo, la propuesta no proporciona caminos de menor coste, ya que aplica una metodología de búsqueda similar al algoritmo DFS pero en un entorno distribuido, y, además, solamente es óptima para el caso con menor diversidad de caminos (k=*2*). De la Oliva [Oliva et al., 2018] también apuesta por un proceso de búsqueda DFS distribuido que obtiene árboles de expansión disjuntos en

enlaces, con el objetivo de prevenir ataques *Man-In-the-Middle* mediante la división de procesos de usuario por las rutas disjuntas. La disjuntividad entre árboles está garantizada, ya que el proceso de búsqueda ejecuta instancias del algoritmo DFS de forma secuencial, eliminando del nuevo proceso de búsqueda los enlaces utilizados en árboles previos. Esta propuesta también incluye mejoras en la construcción de los árboles de expansión, ya que el proceso de búsqueda incluye, además, el cálculo del diámetro del árbol, lo que es útil para construir árboles de expansión de menor longitud y dejar más enlaces libres para futuros árboles, a la vez que acelera el proceso de búsqueda.

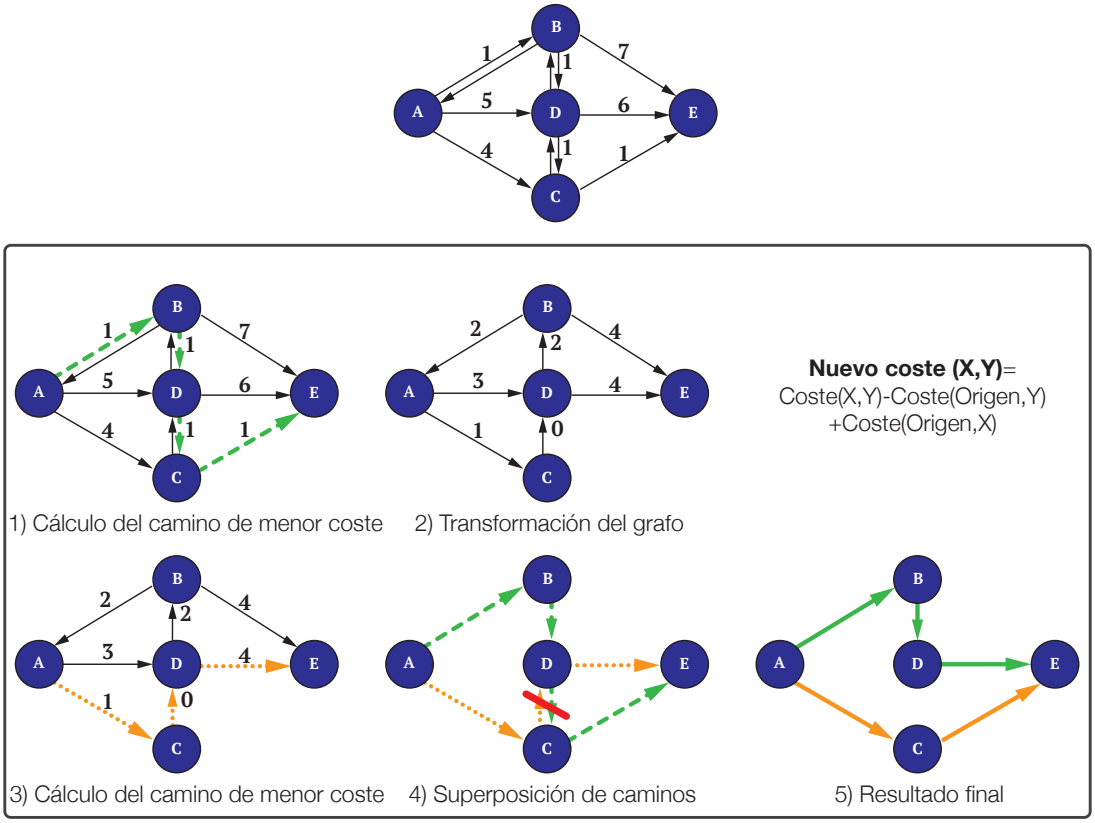
Otras propuestas añaden, además de la disjuntividad entre caminos, el camino más corto o de menor coste como, por ejemplo, la propuesta de Ogier y Shacham [Ogier & Shacham, 1989], quienes diseñan un algoritmo distribuido que obtiene dos árboles disjuntos (en nodos o en enlaces) cuyo coste global es mínimo. [Hadid et al., 2017] también obtienen dos caminos disjuntos (en nodos) entre un par de nodos, diseñando un proceso distribuido espaciado en cuatro fases: obtención del camino de menor longitud aplicando BFS, creación de árboles de mínima longitud con BFS desde los nodos pertenecientes al camino mínimo, estudio de la existencia de caminos disjuntos con los datos obtenidos en las fases previas, y la construcción de los caminos. [Sidhu et al., 1991] también apuesta por un modelo que ofrece caminos disjuntos en nodos sin limitar el número de caminos descubiertos. En su solución, se incluyen árboles disjuntos en nodos, entre los que se encuentra el árbol de mínimo coste. Además, incluye un mecanismo de intercambio de información entre vecinos que incluye datos sobre costes de enlaces y caminos alternativos. [van der Kluit et al., 2017] también patenta un mecanismo de intercambio de mensajes con información topológica de costes, limitando la difusión de los mensajes conforme a tres reglas: reenvío de mensajes solo si no existe información previa sobre el camino o se mejora su coste, prohibición de reenvío a nodos que pertenecen al camino que se está descubriendo, o bloqueo del mensaje si el nodo está conectado directamente con el nodo destino. Además, gracias a este mecanismo de intercambio, la propuesta obtiene múltiples caminos disjuntos entre un par de nodos con coste global mínimo.

[Tanaka, 2014] parte de un camino de menor coste, obtenido a partir de otro protocolo o aplicación, para ofrecer un segundo camino disjunto en enlaces al camino dado. El proceso realiza una exploración de la red mediante una difusión de mensajes (similar a otras explicadas en el apartado de múltiples caminos en entornos distribuidos), excluyendo al camino de menor coste para asegurar la disjuntividad. Este proceso de exploración inunda la red con un paquete de exploración, donde resulta ganador el mensaje que encuentra el camino más rápido y llega antes a destino, obteniendo el camino con menor latencia de la red. Es importante destacar la importancia que tiene el camino de mínima latencia, ya que permite encaminar el tráfico por el camino de menor coste, o redirigir el tráfico a un camino de respaldo en el menor tiempo posible.

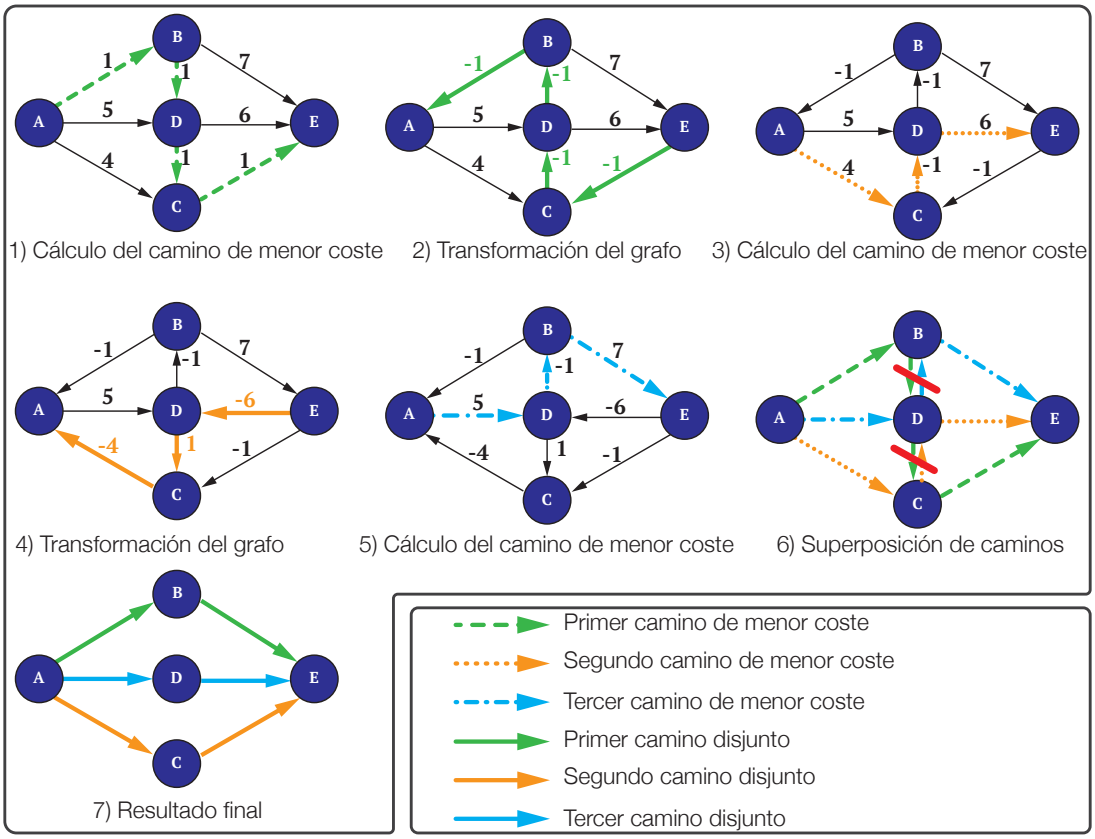
Entornos centralizados

La búsqueda de múltiples caminos disjuntos se remonta al año 1974, cuando [Suurballe, 1974] se adentró en el análisis y la búsqueda de caminos disjuntos en redes de comunicaciones, obteniendo en 1984 un método rápido de búsqueda de caminos disjuntos en grafos ponderados dirigidos y no dirigidos [Suurballe & Tarjan, 1984]. Este algoritmo utiliza el algoritmo de Dijkstra para obtener dos caminos disjuntos en enlaces con coste global mínimo, manteniendo la complejidad computacional en los mismos niveles que Dijkstra. El proceso de búsqueda se estructura en cuatro pasos, siendo el primero de ellos el encargado de obtener el árbol de mínimo coste desde un nodo raíz aplicando, precisamente, el algoritmo de Dijkstra. Este árbol contiene el camino de menor coste entre los nodos origen y destino que se utilizará en la última fase. La segunda fase construye un nuevo grafo eliminando del grafo original el camino de menor coste, al mismo tiempo que actualiza el coste de los enlaces mediante una ecuación que deja con peso nulo los enlaces utilizados por el árbol de mínimo coste, y con peso positivo los enlaces restantes del grafo. El tercer paso vuelve a aplicar Dijkstra sobre el nuevo grafo, obteniendo un nuevo camino de coste mínimo entre los nodos origen y destino. Por último, se superponen los caminos de mínimo coste obtenidos en la primera y en la tercera fase, y se eliminan los enlaces en común, dando lugar a dos caminos disjuntos en enlaces. No obstante, este proceso puede obtener a caminos disjuntos en nodos en función de la topología sobre la que se trabaje. [Bhandari, 1994] diseña un algoritmo más eficiente que simplifica el proceso de búsqueda de Suurballe para obtener múltiples caminos disjuntos en enlaces a costa de incluir pesos negativos en el grafo. El proceso es similar, calculando en primer lugar el camino de menor coste entre los nodos origen y destino y, tras obtener el camino, se invierte tanto el sentido de los enlaces que componen el camino como su coste, para volver a calcular el segundo camino de menor coste con el nuevo grafo. El proceso se repite tantas veces como sea necesario, para, al final, superponer todos los caminos de menor coste y eliminar los enlaces que tengan en común. Para comprender el proceso de búsqueda de ambos protocolos y ver gráficamente sus diferencias, la Figura 2.4 desgrana el proceso de cada propuesta bajo la misma topología. Además, dadas las cualidades morfológicas de la topología, ambas soluciones obtienen caminos disjuntos en nodos.

Las propuestas anteriores sientan las bases sobre las que se construye el problema de los múltiples caminos disjuntos en entornos centralizados, como por ejemplo [Shaikh, 1995], que extiende el algoritmo de Bhandari para separar físicamente rutas en redes ópticas. Esta separación física incluye el uso de equipamiento único en cada camino, con el objetivo de tener un segundo camino de respaldo totalmente independiente del camino principal, y así aumentar la fiabilidad de la red. Este tipo de redes incrementan el ancho de banda de transmisión a la vez que reducen los costes de operación, por lo que su despliegue se ha extendido en estos últimos años. Este incremento del ancho de banda de transmisión se debe al uso de la luz para transportar la información, unido a



a) Algoritmo de Suurballe.



b) Algoritmo de Bhandari. c) Leyenda.

Figura 2.4: Ejemplo de los algoritmos de Suurballe y Bhandari.

la multiplexación de portadoras ópticas que transmiten datos en paralelo. Sin embargo, el cambio de portadora dentro de un camino introduce retardos indeseados y requiere equipos hardware costosos que incrementan notablemente el coste de la infraestructura necesaria [Ahmad, 2002], por lo que el problema de la búsqueda de caminos disjuntos en redes ópticas introduce una incógnita más al problema: cada camino disjunto tiene que asegurar la continuidad de longitud de onda a lo largo de su recorrido. [Manohar et al., 2002] afrontan el problema de la búsqueda de caminos disjuntos en enlaces con asignación de longitud de onda continua, en un proceso dividido en dos fases. En primer lugar, se aplica un algoritmo que, secuencialmente, calcula el camino de menor coste, eliminando los enlaces utilizados por caminos previos para garantizar la disjuntividad. En segundo lugar, partiendo de los caminos físicos descubiertos, se asigna la longitud de onda que cumpla con los requisitos de los enlaces comprendidos en el camino. [Prakash et al., 2019], sin embargo, apuestan por modificar el grafo de la red, adaptándolo a las condiciones de continuidad, y luego aplicar el algoritmo de Suurballe para obtener los caminos disjuntos. La propuesta parte de un grafo ponderado y no dirigido que es convertido en un grafo ponderado y dirigido mediante el etiquetado de los enlaces con restricciones de capacidad y disponibilidad, y su posterior transformación en nodos. Además, también transforma las conexiones internas entre los puertos de cada nodo en enlaces y penaliza los puertos de regeneración de señales ópticas para evitar su uso a lo largo de los caminos. Sobre este nuevo grafo es sobre el que se aplica el algoritmo de Suurballe, que proporciona caminos disjuntos direccionales dadas las condiciones del grafo. Sin embargo, estas restricciones se eliminan al final, basándose en las condiciones del grafo original (ponderado y no dirigido) para obtener caminos bidireccionales.

El algoritmo de Bhandari también se combina con técnicas QoS para aprovechar más eficientemente los recursos de la red. DIMCRA [Guo et al., 2003] obtiene múltiples caminos disjuntos con políticas QoS modificando el algoritmo de Bhandari. Sin embargo, esta modificación puede provocar la aparición de bucles al hacer la inversión de caminos y costes que aplica el algoritmo de Bhandari. La solución pasa por invertir los enlaces, al igual que Bhandari, pero en lugar de invertir su coste, se les asignan costes nulos. También hay que destacar que la solución devuelta puede que no sea óptima en términos de coste global, a cambio de obtener caminos disjuntos. El algoritmo de Dijkstra también se combina con políticas QoS para obtener caminos disjuntos en enlaces y encaminar el tráfico a través de ellos, como por ejemplo en [Abe et al., 2015]. La obtención de los caminos disjuntos se basa en la modificación del grafo topológico y en la ejecución reiterada del algoritmo de Dijkstra. Los costes de los enlaces del grafo se adecuan a las características QoS que demanda el servicio, para ejecutar el algoritmo de Dijkstra y obtener el camino de menor coste entre los nodos origen y destino. Después, se aplica una penalización de coste al camino encontrado en el sentido de la búsqueda (de origen a destino), y se elimina dicho camino en sentido inverso, para evitar que nuevas búsquedas obtengan enlaces que pertenecen a dicho camino. Además, se aplica un factor de penalización multiplicativo entre procesos

de búsqueda, evitando que la duplicidad de caminos entre caminos obtenidos en distintas etapas. No obstante, la propuesta deja abierto el procedimiento, pudiendo aplicar inversión de costes en el proceso de penalización y utilizar el algoritmo Bhandari en lugar del algoritmo de Dijkstra.

[Zuo et al., 2019] también utilizan el algoritmo de Dijkstra para obtener caminos disjuntos (en nodos o en enlaces) en en un proceso que combina el algoritmo de Dijkstra con un heurístico propio. La propuesta calcula recursivamente los caminos de menor coste entre un par de nodos origen-destino con el algoritmo de Dijkstra, eliminando del grafo los caminos descubiertos y asegurando así la disjuntividad. Después, aplica un heurístico que encuentra los caminos del conjunto anterior que satisfacen los requisitos de QoS. [Doshi & Kamdar, 2018] también utilizan un procedimiento estructurado en dos fases aplicando el algoritmo de Dijkstra para obtener caminos disjuntos en enlaces. Análogamente al trabajo anterior, la primera fase obtiene múltiples caminos entre un par de nodos origen-destino y, después, la segunda fase, aplica un heurístico de optimización para seleccionar aquellos caminos que cumplen con los criterios QoS. [Tolba, 2017] realiza el proceso a la inversa. En primer lugar, el algoritmo divide el grafo original en subgrafos de menor tamaño y, después, aplica el algoritmo de Dijkstra desde los nodos que dividen el grafo hacia los nodos origen y destino. Al combinar las soluciones se obtienen caminos parcialmente disjuntos, ya que los caminos comparten, además de los nodos extremos, los nodos que dividen el grafo.

MADSWIP [Taft-Plotkin et al., 1999] descompone el proceso de búsqueda de caminos disjuntos en cuatro fases inspirándose en el algoritmo de Suurballe. Además, la propuesta no se limita a buscar caminos entre un par de nodos, sino que busca caminos disjuntos entre un nodo origen y el resto de nodos de la topología, y cuando no es posible obtenerlos, devuelve caminos parcialmente disjuntos. La primera fase aplica una variante del algoritmo de Dijkstra para obtener el árbol de menor coste, atendiendo a los requisitos de ancho de banda y longitud del camino que fue diseñado por [Wang & Crowcroft, 1996]. En segundo lugar, se modifican los costes del grafo, inspirándose en el algoritmo de Suurballe, pero añadiendo una métrica adicional al coste del enlace, que selecciona el ancho de banda mínimo entre el ancho de banda del enlace y el ancho de banda del camino. La tercera fase reetiqueta los nodos atendiendo a los nuevos costes obtenidos en la segunda fase, y por último, la cuarta fase construye los caminos desde los nodos destinos hacia el nodo raíz en función del nuevo etiquetado.

Los trabajos que aplican políticas de QoS analizados hasta el momento utilizan un algoritmo de búsqueda clásico como Dijkstra, Bhandari o Suurballe para obtener los caminos disjuntos. Por el contrario, [Lemeshko et al., 2019] apuestan por un enfoque completamente distinto, sustituyendo los algoritmos clásicos por una función de optimización que obtiene los caminos disjuntos en función de unos criterios previamente definidos. [Orda & Sprintson, 2004] también apuestan por un enfoque similar que utiliza funciones de optimización para resolver el problema de la búsqueda de caminos disjuntos.

Otras de las aplicaciones típicas en las que se emplean múltiples caminos disjuntos es el encaminamiento a través del camino o caminos más corto, sin atender a políticas de QoS. [Tapolcai et al., 2015, Tapolcai et al., 2019] apuestan por ofrecer un algoritmo eficiente que obtiene árboles disjuntos de longitud mínima en grafos no dirigidos y ponderados. En primer lugar, aplica el algoritmo de Suurballe entre el nodo raíz y el resto de los nodos del grafo, para obtener caminos disjuntos entre dichos pares de nodos y, después, ordena los nodos de menor a mayor distancia desde el nodo raíz. Una vez ordenados los nodos, el algoritmo recorre dicha lista añadiendo, por cada nodo seleccionado, el enlace hacia el siguiente nodo vecino de la solución obtenida por el algoritmo de Suurballe. No obstante, para garantizar la disjuntividad entre árboles, se seleccionan enlaces en sentidos opuestos (del primer camino se selecciona el enlace que va en sentido opuesto al nodo raíz y del segundo camino el enlace en dirección contraria). Cuando se recorren todos los nodos se obtienen dos árboles disjuntos en enlaces, sin embargo, la solución no proporciona disjuntividad física total, ya que ambos árboles pueden confluir en un mismo enlace, pero en sentidos opuestos. La propuesta puede ser interesante para escenarios donde la disjuntividad física no sea un requisito imprescindible.

Los múltiples caminos disjuntos también pueden utilizarse para mejorar la seguridad y la fiabilidad de la red. Por ejemplo, [Mohan et al., 2018] utilizan los múltiples caminos disjuntos para mejorar la fiabilidad del plano de control In-Band en redes SDN. Al igual que otras propuestas anteriores, esta solución apuesta por un heurístico propio que obtiene dos caminos disjuntos en nodos, detecta ataques maliciosos, establece un mínimo de calidad de transmisión y minimiza la longitud de los caminos. A todas esas variables les aplica una función de optimización que encuentra la solución con un tiempo de resolución razonable en redes de pequeña escala. [Nguyen et al., 2017] tienen por objetivo transmitir la información de un mismo flujo de datos entre varios caminos disjuntos para evitar ataques *Man-In-the-Middle*. Apuesta por un esquema basado en algoritmos de búsqueda en profundidad, como DFS, que eliminan los caminos que se van descubriendo. Después aplica un filtro que selecciona solamente caminos disjuntos en nodos.

A continuación se muestran otras propuestas que también suscitan interés, pero que no encajan en ninguna de las aplicaciones descritas en este apartado. [Martín et al., 2020] hacen una apuesta firme por el uso de algoritmos evolutivos para descubrir caminos disjuntos en enlaces. Estos algoritmos contienen métodos de búsqueda y optimización inspirados en la teoría de la evolución Darwiniana, que parten de una población conocida (repesentando posibles soluciones del problema) y se mezclan entre sí para que compitan y muten. Las soluciones óptimas del problema, son aquellas mutaciones que consiguen mayor puntuación. En este caso particular, la población inicial consiste en un conjunto de caminos de mínimo coste entre los nodos origen y destino, obtenidos a partir de varias ejecuciones del algoritmo de Dijkstra. Con la población inicial y las ecuaciones que definen el problema, se aplican los algoritmos evolutivos que obtienen la solución final dada.

Existen otros enfoques que apuestan por el uso de algoritmos inspirados en

comportamientos animales para obtener dos caminos disjuntos como, por ejemplo, los que imitan el comportamiento de las hormigas en su proceso de búsqueda de alimentos [Colorni et al., 1991]. Las hormigas exploran un área de terreno desde el hormiguero en busca de comida, dejando un rastro de feromonas allá por donde pasan. Cuando encuentran la comida, la hormiga que descubre el camino más corto llegará primero, por lo que las demás hormigas solo tienen que seguir el rastro de feromonas marcado por la hormiga exploradora. Los caminos de menor coste/longitud son más transitados por las hormigas, por lo que el rastro de feromonas en dicho camino se ve incrementado con respecto a otros posibles caminos. Al final, el camino más corto es el que mayor concentración de feromonas contiene y el que guía a las hormigas hasta la fuente de alimentos. Mediante un algoritmo inspirado en las colonias de hormigas, [Xingyu et al., 2018] obtienen un conjunto de caminos de longitud mínima entre los nodos origen y destino, aplicando sobre dicha solución, posteriormente, algoritmos evolutivos para ofrecer una solución final que cuenta con dos caminos disjuntos. También ofrece una variante alternativa que consiste en aplicar un algoritmo de hormigas para obtener el camino de menor coste y después aplicar el algoritmo de Bhandari para obtener el segundo camino disjunto, sin embargo, esta variante es menos óptima que la primera.

Otro comportamiento animal en el que se inspiran los algoritmos de optimización está relacionado con los enjambres de partículas (similar a las abejas, por ejemplo), en el que las partículas se mueven dentro de un área de soluciones y se acercan hacia las partículas con mayor aptitud. [Al-Turjman & Alturjman, 2020] utilizan este enfoque para obtener caminos disjuntos en nodos. La propuesta introduce la formulación del problema incluyendo el grafo de la red, junto con restricciones de consumo energético, ancho, retardo, etc. y la definición de caminos disjuntos para aplicar el algoritmo de optimización de partículas y obtener los resultados esperados.

Capítulo 3

Planteamiento del problema

Este capítulo enumera las carencias encontradas a lo largo del capítulo del estado del arte para sentar las bases sobre las que construir la presente Tesis Doctoral.

En primer lugar, se ha visto que las redes SDN In-Band reducen los costes de despliegue y mantenimiento, al aprovechar el despliegue de la capa de infraestructura para encaminar tanto el tráfico de control como el tráfico de datos, evitando así el despliegue paralelo de una red dedicada exclusivamente al tráfico de control. Sin embargo, el modelo SDN In-Band no cuenta con un estándar de facto que defina la creación de rutas entre el controlador y los switches a través de la capa de infraestructura, por lo que existen varias alternativas disponibles en la literatura. Sin embargo, estas propuestas no se han consolidado como una apuesta firme, dejando la puerta abierta a nuevos trabajos. Además, dentro del estudio realizado se han detectado las siguientes carencias:

- La ausencia de un protocolo de comunicación In-Band rápido y autoconfigurable. En el estado del arte es frecuente encontrar propuestas que necesitan una configuración manual por parte del administrador de la red para establecer el canal de control en redes SDN In-Band, lo que resulta inoperante para redes de gran escala. Entre las actuaciones que realiza el administrador se encuentra la configuración de reglas de reenvío prioritarias para priorizar el tráfico de control in-band, o la introducción manual de direcciones IP en todos los switches, necesarias para establecer la comunicación con el controlador. No obstante, también existen otros trabajos que solucionan este handicap, diseñando protocolos que realizan la conexión de forma automática. Sin embargo, estas últimas propuestas son excesivamente lentas estableciendo las conexiones entre el controlador y los switches, debido al uso de protocolos externos como STP o RSTP para crear el árbol de expansión que comunica al controlador con todos los switches. Además, la operativa de estos protocolos deshabilita los enlaces que no utiliza el árbol de expansión, lo que reduce la cantidad de recursos disponibles para encaminar el tráfico e incrementa la probabilidad de congestión.
- La falta de un sistema de recuperación ante fallos de red rápido y escalable. A

menudo, los trabajos estudiados bien utilizan esquemas de protección 1:1 que reservan a priori caminos de respaldo, bien recurren a sistemas de recuperación reactivos que buscan rutas alternativas tras detectar el fallo. Mientras que los esquemas de protección, en general, garantizan un tiempo de recuperación dentro del margen de 50ms establecido por [Niven-Jenkins et al., 2009], no son sistemas que escalen bien, ya que necesitan mucho tiempo de cómputo y guardar mucho estado para mantener una ruta de respaldo entre el controlador y cada uno de los switches bajo su dominio. Los esquemas reactivos reducen considerablemente estos inconvenientes al recalcular las rutas en base al tramo o tramos afectados por el fallo. Sin embargo, el tiempo invertido en obtener el camino de respaldo supera en muchas ocasiones el límite de 50ms, ya que se tiene que detectar el fallo y buscar una ruta alternativa.

En segundo lugar, se han visto las ventajas que aportan los enfoques híbridos en redes SDN:

- Se reduce la carga computacional de controlador al delegar las tareas al plano de datos, quedando libre el controlador para otras tareas de mayor prioridad.
- Se incrementa la robustez de la red al distribuir procesos por la capa de infraestructura. Esta es una de las ventajas que aportan los sistemas distribuidos, ya que la cooperación entre dispositivos puede absorber fallos parciales del sistema.
- Se garantiza un crecimiento escalable de la red. En una red SDN la incorporación de nuevos dispositivos supone una carga adicional de trabajo al controlador, quien tiene que atender, procesar y responder las peticiones de un nuevo dispositivo, lo que implica un aumento de los requisitos hardware del controlador y por ende, de los costes de infraestructura. Sin embargo, en una red SDN híbrida muchas de las funciones se realizan en el plano de datos, distribuyendo la carga entre todos los dispositivos y liberándolo de ella al controlador. Además, cada dispositivo maneja información a nivel local, lo que no incrementa en exceso los requisitos hardware de los switches.

No obstante, a pesar de las ventajas que aportan las redes híbridas, no son muchas los trabajos que apuestan por este tipo de enfoques debido a la complejidad de desarrollo que introduce una computación distribuida, donde además de adaptar los procesos para trabajar, precisamente, de forma distribuida, se deben incluir mecanismos de intercambio de información entre vecinos.

Por último, se han enumerado las ventajas que aporta el cálculo de múltiples caminos en redes de comunicaciones (optimización de recursos, e incremento de la seguridad y fiabilidad en la transmisión de datos), viéndose éstas reforzadas si los caminos tienen la propiedad adicional de disjuntividad. Además, se analizaron tanto propuestas que trabajan de forma centralizada, como propuestas que cooperan de forma distribuida, con el objetivo

de tener una visión clara de ambos enfoques. Sin embargo, entre las propuestas analizadas se han encontrado las siguientes debilidades:

- La diversidad de caminos frecuentemente está limitada a descubrir dos caminos entre un par de nodos, suficiente para labores de respaldo básico, pero insuficiente si se quieren proporcionar beneficios adicionales.
- Se encuentran propuestas que no aportan en su solución disjuntividad entre caminos, o la aportan parcialmente, lo que reduce los beneficios de la multiplicidad de caminos. Además, los trabajos suelen limitar la búsqueda de caminos disjuntos a una de sus variantes, caminos disjuntos en enlaces o caminos disjuntos en nodos, limitando la aplicabilidad de las soluciones.
- Los entornos centralizados basan el proceso de búsqueda en algoritmos conocidos, modificándolos o añadiéndoles etapas nuevas para adaptarlos a los nuevos retos. Sin embargo, la escalabilidad de estas propuestas sigue siendo su punto débil, ya que en redes de gran tamaño la operativa matemática necesaria para resolver el problema requiere una capacidad y tiempo de cómputo considerables.
- En los entornos distribuidos sí que se garantiza la escalabilidad, sin embargo, las propuestas no están adaptadas a los nuevos retos que demandan las redes de comunicaciones actuales como, por ejemplo, las redes SDN híbridas.

Por todo ello, la presente Tesis Doctoral se adentra en el mundo de las redes SDN en combinación con el problema del multicamino, pretendiendo superar los siguientes retos:

- Encontrar un protocolo autoconfigurable, rápido y escalable que proporcione las rutas para establecer la comunicación del plano de control en un modelo de red SDN In-Band. Además, esta propuesta debe contener un mecanismo de recuperación ante fallos que recupere fallos de red en menos de 50ms. Al trabajar en una red SDN In-Band, el protocolo debe trabajar de forma distribuida e independiente del controlador, debiendo recibir éste únicamente la información de las rutas una vez que se han calculado. Asimismo, la propuesta debe encontrar múltiples rutas para garantizar un sistema de recuperación rápido, pero al mismo tiempo no debe desperdiciar recursos de red reservando recursos del sistema.
- Diseñar protocolos multicamino compatibles con redes SDN híbridas que no limiten la diversidad de caminos, que apuesten por la disjuntividad entre caminos, y ofrezcan tanto caminos disjuntos en enlaces, como caminos disjuntos en nodos. Se apuesta por una operación distribuida en la capa de infraestructura en la que los switches de la red cooperan para encontrar los múltiples caminos disjuntos. Tras encontrar los caminos, los switches devuelven el resultado al controlador de la red para que los utilice. Además, la propuesta debe contar con procesos simples e intercambios de información entre vecinos que garanticen la escalabilidad del sistema.

- Aprovechar los conocimientos adquiridos en las redes SDN híbridas sobre el problema del multicamino para crear una algoritmo centralizado que calcule múltiples caminos disjuntos, ya que no es siempre posible utilizar redes SDN híbridas. Además, para potenciar su implantación, su diseño debe agnóstico, con el objetivo de que pueda ser utilizado tanto en el ámbito de las redes de comunicaciones como en otras disciplinas.

Capítulo 4

Amaru

En este capítulo se presenta la primera aportación de la Tesis, Amaru, un protocolo de encaminamiento In-Band multicamino, autoconfigurable, escalable y rápido en la obtención de los caminos, que incrementa la robustez en los procesos de comunicación del plano de control, proporcionando, además, recuperaciones ante fallos de la red en menos de 50ms. El origen del nombre hace referencia al dragón de Amaru [Steele & Allen, 2004], animal mitológico cuyo cuerpo tiene forma de serpiente alada, similar a la trayectoria serpenteante que tienen los caminos descubiertos por el protocolo. Amaru explora todos los posibles caminos entre el dispositivo/switch SDN de la capa de infraestructura conectado directamente con el controlador y el resto de dispositivos/switches SDN de la red. Para ello, incorpora un sencillo mecanismo de difusión de información que trabaja de forma distribuida y no afecta al rendimiento de la red, garantizando en todo momento la escalabilidad. Además, Amaru proporciona un sistema de recuperación ante fallos de red rápido y ágil, gracias a que aprovecha los múltiples caminos descubiertos para utilizarlos como rutas de respaldo.

Un escenario de aplicación típico para el protocolo Amaru son las redes de proveedores de servicios de telecomunicaciones que abarcan grandes extensiones de terreno. En este entorno encaja a la perfección un modelo SDN In-Band que reaprovecha el equipamiento de red destinado a la capa de infraestructura para el encaminamiento del tráfico de control, evitando así el despliegue de una red paralela destinada exclusivamente al tráfico de control. Una implementación real de este formato es Rogers Fiber Backbone [Rogers Communications Inc., 2018], una red ISP que conecta de oeste a este Canadá y parte de Estados Unidos a través de un modelo SDN In-Band. La Figura 4.1 muestra gráficamente la topología Rogers, siendo Toronto la ciudad elegida para ubicar el controlador SDN. Desde Toronto, el controlador gestiona la red a través de la única infraestructura física desplegada, utilizando para ello un esquema SDN In-Band. Todas estas características hacen que la topología Rogers sea utilizada más adelante como escenario de pruebas sobre el que ejecutar Amaru.

El capítulo comienza mostrando las bases sobre las que se construye Amaru

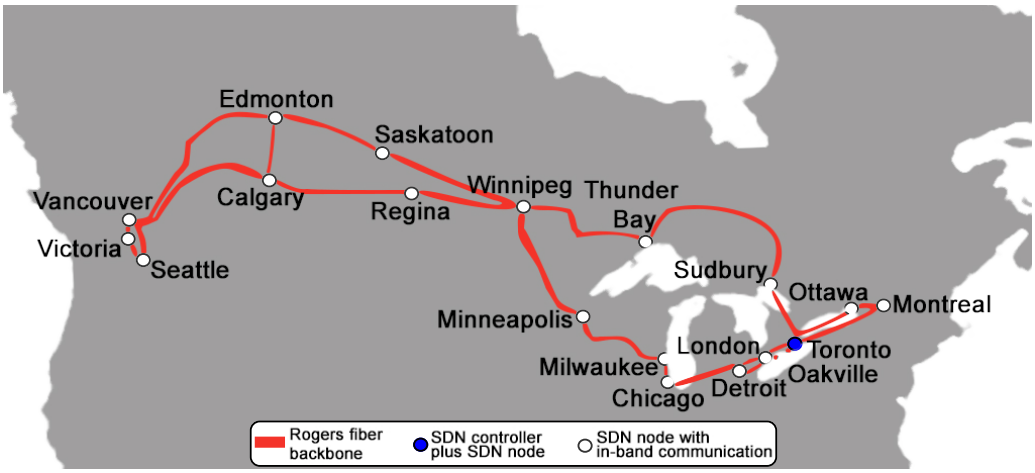


Figura 4.1: Topología Rogers Fiber Backbone.

para, posteriormente, describir con detalle su operativa. Por último, se mostrará la implementación en distintas plataformas y la evaluación en varios escenarios.

4.1 Bases

El proceso de búsqueda de rutas para el encaminamiento In-Band de Amaru se basa en dos procedimientos desarrollados por el equipo investigador NetServ, dentro del grupo Networks and Intelligent Systems (NetIS) de la Universidad de Alcalá, destinados al desarrollo de técnicas de exploración de red con inundación de mensajes libres de bucles, y a la asignación automática de direcciones en redes jerárquicas de centros de datos.

La familia de protocolos All-Path [Ibáñez & Rojas, 2013,Rojas et al., 2011,Rojas et al., 2015] está compuesta por un conjunto de protocolos distribuidos que descubren el camino de mínima latencia entre dos switches, mediante un envío controlado de mensajes. El proceso se estructura en dos fases, no necesita conocer la topología de red subyacente para descubrir el camino de mínima latencia, y aprovecha el intercambio de mensajes propio del protocolo ARP para descubrir los caminos. La primera fase consiste realizar una exploración de la red a través de una difusión controlada del mensaje ARP Request, en la cual, los switches anotan la interfaz de entrada del primer mensaje de exploración, y después reenvían dicho paquete a través de todas sus interfaces, excepto por la interfaz de entrada. Además, copias tardías del mensaje de exploración son rechazadas para evitar tormentas de mensajes. La segunda fase arranca con la respuesta al paquete ARP Request, un paquete ARP Reply en sentido opuesto (hacia el switch origen), enviado desde el dispositivo destino. Los switches que reciben el paquete ARP Reply anotan la interfaz de entrada de dicho paquete, y lo reenvían a través de la interfaz de entrada anotada en la primera fase. De esta forma, cada switch tiene caracterizado el camino en los dos sentidos. El proceso finaliza cuando el mensaje ARP Reply alcanza al switch origen, quedando establecido el camino de mínima latencia que comunica a los dispositivos origen

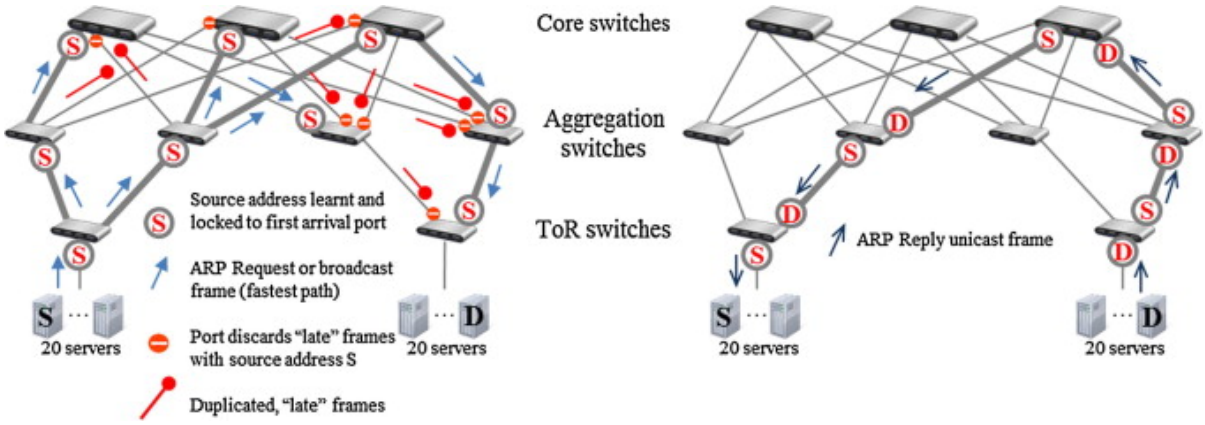


Figura 4.2: Ejemplo del protocolo ARP-Path perteneciente a la familia All-Path. Fuente: [Rojas et al., 2015].

y destino. La Figura 4.2 muestra gráficamente este proceso en una red compuesta por 9 switches y 40 servidores. El servidor origen, S, envía el mensaje ARP Request para conocer la dirección IP del switch destino. Este mensaje atraviesa la red conforme a la política explicada anteriormente hasta que alcanza el servidor destino (D). D responde con un mensaje ARP Reply, que llega hasta el switch origen a través de la información recogida en la fase de exploración, creando un camino bidireccional entre los servidores S y D.

GA3 [Rojas et al., 2017] es un protocolo de asignación de direcciones en redes jerárquicas de centros de datos que utiliza la técnica de inundación controlada de la familia All-Path para distribuir la asignación de direcciones. GA3 introduce un mecanismo de difusión de tramas de direccionamiento que arranca en los nodos marcados como raíz, y que asigna direcciones a dispositivos de jerarquía inferior, añadiendo a la dirección base la identidad del puerto de salida. La difusión de las direcciones se hace a través del mecanismo de exploración provisto por la familia All-Path, mediante el reenvío de las tramas por todas las interfaces del switch menos por la interfaz en la que se recibió el mensaje, solo que ahora, además, por cada paquete de direccionamiento reenviado por

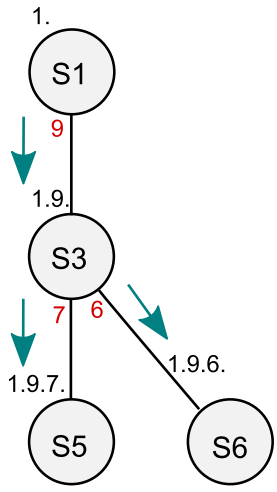


Figura 4.3: Ejemplo de GA3 en red jerárquica. Fuente: [Rojas et al., 2017].

una interfaz de salida, se añade la identidad de dicho puerto a la dirección incluida en el paquete. La Figura 4.3 muestra un ejemplo del proceso compuesto por una red jerárquica de cuatro nodos en la que S1 es el nodo raíz cuyo identificador es *1.*. S1 difunde el mensaje de direccionamiento incluyendo en el mensaje su identificador y la identidad del puerto de salida, dando lugar a la dirección *1.9.*. S3 recibe el mensaje con la dirección *1.9.* y se la guarda, puesto que está en un nivel inferior de la jerarquía, repitiendo el proceso de reenvío a través de todas sus interfaces excepto por la interfaz de entrada del mensaje. El proceso da lugar a las direcciones *1.9.6.* y *1.9.7.* que son recibidas y aceptadas por los switches S6 y S5 respectivamente al pertenecer a un nivel inferior de la jerarquía.

Amaru, para descubrir los múltiples caminos que comunican al controlador SDN con todos los dispositivos de la capa de infraestructura bajo un esquema In-Band, incluye un sistema de etiquetado que incorpora mecanismos de asignación y distribución de direcciones similar al propuesto en GA3. Sin embargo, en Amaru todos los dispositivos de la capa de infraestructura se encuentran en el mismo nivel de la jerarquía, por lo que el proceso cambia ligeramente para eliminar las restricciones jerárquicas impuestas por GA3.

4.2 Proceso de asignación de direcciones y descubrimiento de rutas de Amaru

Una vez vistos los protocolos en los que se inspira Amaru, se describe la operativa del protocolo para descubrir las múltiples rutas existentes entre el controlador y los dispositivos de la capa de infraestructura. En primer lugar, la sección presenta el procedimiento de etiquetado de Amaru, mostrando el sistema de identificación y etiquetado y, posteriormente, se detalla el procedimiento seguido para descubrir las múltiples rutas mediante la difusión controlada de etiquetas.

4.2.1 Sistema de identificación/direccionamiento

Amaru emplea un sistema de etiquetado secuencial, en el que se incrementa paulativamente el tamaño de la información contenida en la etiqueta conforme se añaden switches al camino.

La Figura 4.4a muestra un sistema de etiquetado clásico, en el que cada nodo dispone de un identificador único (X, Y y Z), mientras que la Figura 4.4b muestra el sistema de etiquetado de Amaru, el cual añade información por cada switch atravesado desde un switch raíz (X es el switch raíz a partir del cual se construyen el resto de etiquetas). Este sistema de etiquetado secuencial, además de identificar inequívocamente a los switches, permite representar o conocer la conectividad entre ellos. Por ejemplo, el switch *X.Y.Z* sabe que tiene conexión con el switch *X* a través del switch *X.Y*, y

a)Sistema de etiquetado convencional. b)Sistema de etiquetado utilizado por Amaru.

Figura 4.4: Ejemplo de identificación de nodos mediante dos modelos de etiquetado.

conexión directa con el switch *Y*. Este sistema de etiquetado será útil para que los dispositivos de la capa de infraestructura conozcan la ruta hasta el switch raíz que está conectado directamente con el controlador. Además, Amaru introduce un sistema de identificación secuencial que permite descubrir múltiples rutas entre el controlador y cada uno de los switches de la capa de infraestructura. Dicho sistema incluye un direccionamiento especial basado en etiquetas que sustituye al direccionamiento clásico con direcciones MAC utilizado como estándar de direccionamiento en las redes de switches. De hecho, el sistema de direccionamiento/identificación definido por Amaru no modifica la trama estándar de Ethernet definida por el IEEE 802.3 [IEEE, 2018], sino que crea un nuevo modelo de identificación compuesto por etiquetas denominadas Amaru MAC (AMAC), alternativo al direccionamiento estándar. Una etiqueta AMAC consiste en una estructura de datos compuesta por una dirección secuencial que sigue el formato de la Figura 4.4b. El primer elemento de la etiqueta AMAC es el identificador único asignado al controlador, mientras que los siguientes elementos contienen información de los dispositivos atravesados, proporcionando así un sistema de encaminamiento que provee a cada dispositivo una ruta para llegar hasta el controlador. No obstante, no existe una asignación de AMAC única por switch, sino que el proceso de generación y propagación de las AMACs produce múltiples direcciones diferentes para un mismo dispositivo, ofreciendo así rutas alternativas que pueden ser útiles para tareas de respaldo en caso de fallo en la ruta principal.

La estructura de las AMAC tiene el mismo tamaño que una dirección MAC (48 *bits*), con el objetivo de hacer una sustitución directa MAC-AMAC. Sin embargo, de los 48 *bits* que ocupa, solo son utilizables 46, puesto que, al igual que en el estándar de direccionamiento MAC [International Organization for Standardization, 1991], los dos primeros *bits* están reservados. El primer *bit* se reserva para la gestión de direcciones multicast (que incluye la dirección de broadcast), mientras el segundo indica si la dirección está administrada de forma local o es universal. Como las AMAC son creadas y administradas localmente, este segundo *bit* estará siempre habilitado. No obstante, a pesar de contar con un tamaño fijo, la estructura interna de las AMAC es variable, pudiendo fijar el número máximo de saltos (nivel de profundidad) que tiene una AMAC, lo que permite obtener 9 niveles de profundidad con 5 *bits* por nivel (y sobra 1 *bit*), u 11 niveles de profundidad con 4 *bits* por nivel, etc. Esta variabilidad en el campo de las AMAC permitirá más adelante estudiar el comportamiento de Amaru bajo diversos criterios, con diferentes niveles de profundidad. A modo resumen, la Figura 4.5 muestra la estructura

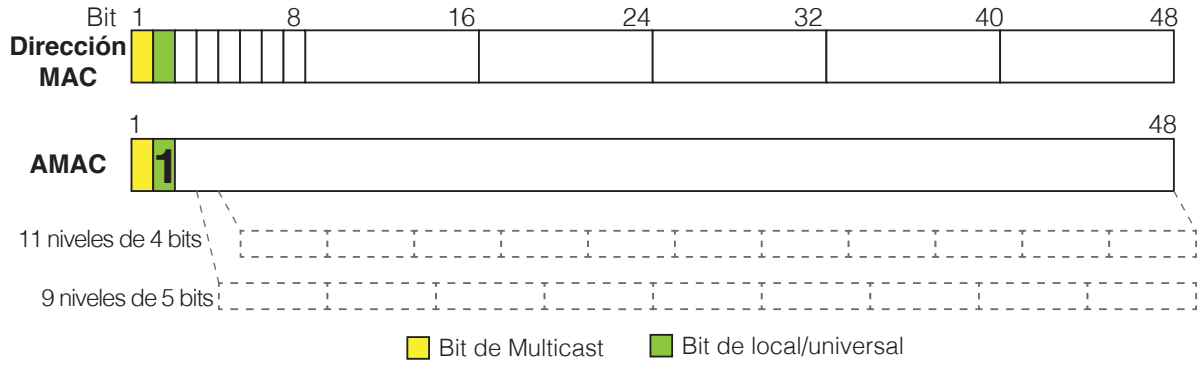


Figura 4.5: Estructura del etiquetado AMAC frente a una dirección MAC convencional.

de una dirección MAC convencional frente a las etiquetas AMAC.

El motivo de crear un sistema de identificación dinámico basado en etiquetas se debe a que en las redes SDN los identificadores de los dispositivos de la capa de infraestructura habitualmente son asignados, bien por el controlador SDN, bien por el administrador de la red, frente a los sistemas tradicionales en los que la identificación del dispositivo viene predeterminada de fábrica. Este dinamismo ha permitido crear un sistema de identificación activo completamente autónomo que, al mismo tiempo que identifica a los switches, proporciona información de encaminamiento. Asimismo, se ha optado por un sistema que aprovecha la estructura de la trama Ethernet para sustituir las direcciones MAC de un paquete por etiquetas AMACs, proporcionando un sistema eficaz que no requiere modificaciones en la estructura de datos de las tramas enviadas por los switches.

De hecho, el uso de sistemas de etiquetado con un esquema de direccionamiento MAC local está incluido en la corrección del estándar 802.c del año 2017 [IEEE, 2017]. En él se proponen estructuras adicionales y rangos de direcciones MAC reservados para su uso en diversas aplicaciones, por lo que el esquema de direccionamiento de Amaru encaja con la filosofía de direccionamiento MAC.

4.2.2 Propagación y asignación de etiquetas/direcciones

Una vez visto el sistema de identificación/etiquetado utilizado por Amaru, así como la descripción de su estructura, esta sección se centra en el método de propagación de dichas etiquetas y su asignación a los dispositivos de la capa de infraestructura. Como se ha comentado anteriormente, el modelo de etiquetado de Amaru permite que un mismo dispositivo tenga varias etiquetas AMAC que, además de identificar unívocamente a cada uno de los dispositivos de la capa de infraestructura, le proporciona una ruta hacia al controlador, obteniendo así múltiples rutas de respaldo en caso de que el canal de control sufra algún fallo. Estas etiquetas múltiples se generan aplicando las técnicas de exploración descritas en la Sección 4.1, junto con un etiquetado secuencial en el que los nodos atravesados por las tramas de explotación van añadiendo información a la etiqueta. La Figura 4.6 muestra el escenario de ejemplo sobre el que se escenifica gráficamente

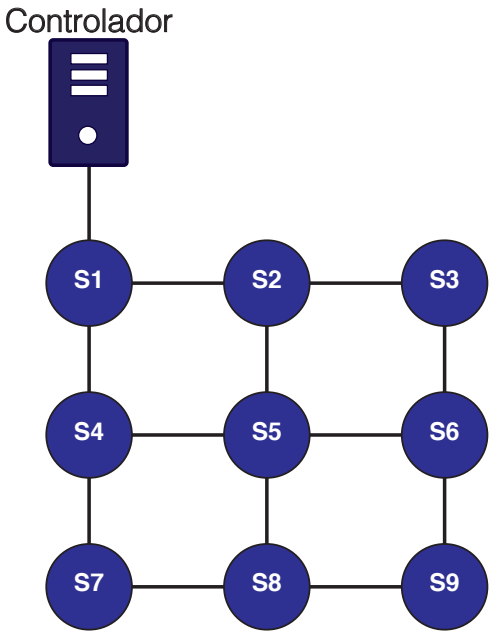


Figura 4.6: Escenario de ejemplo.

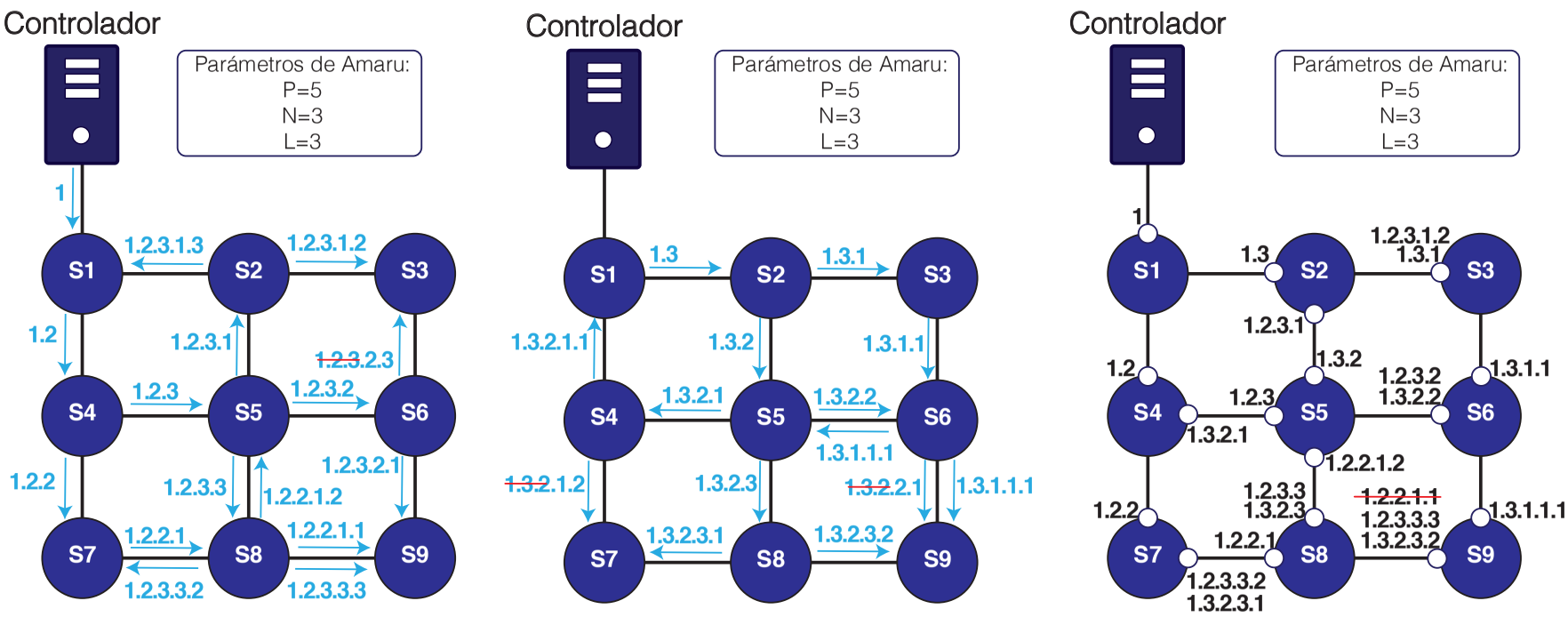
la operativa de Amaru. Dicha red está compuesta por nueve switches y un controlador conectado directamente a uno de ellos (*S1*), donde además, se implementa un esquema In-Band para realizar la comunicación del plano de control entre el controlador y los switches.

El proceso de propagación y asignación de etiquetas comienza en el controlador, generando una trama de exploración que recorrerá toda la red. Dicha trama incluye una etiqueta con un valor inicial que identifica al controlador, conocida como Amaru Root (ARoot). Tras enviar la trama a la red, ésta recopila información por cada switch atravesado, añadiéndola a la AMAC contenida en la trama. Atravesar un switch implica estar un paso más alejado del controlador y añadir un nivel de profundidad addicional a la etiqueta AMAC, por lo que, cuando una etiqueta alcance el nivel de profundidad máximo fijado, se detendrá su difusión por la red (en los ejemplos de la Sección 4.2.1 se han fijado dos situaciones con una profundidad de 9 y 11 saltos, pero pueden ser menor). No obstante, también existen otras situaciones que impiden el aprendizaje o la difusión de las tramas de exploración: el número máximo de direcciones AMAC que puede aprender un switch, y el número de elementos que pueden compartir dos AMAC; lo que permite acotar el número máximo de saltos que puede tener un camino, el número máximo de caminos múltiples por switch, y el número de elementos máximo que pueden compartir los múltiples caminos. Estas tres restricciones se han parametrizado en tres variables denominadas P (profundidad máxima de las etiquetas), N (número máximo de etiquetas AMAC aprendidas en cada dispositivo) y L (longitud máxima que pueden compartir las etiquetas AMAC en un mismo dispositivo). La decisión de limitar la propagación de AMACs no es casual, ya que si no se establecen unas reglas para limitar su difusión, el número de caminos posibles puede ascender fácilmente al rango de millones en redes

de solo doce nodos [Roberts & Kroese, 2007]. Un número tan elevado de caminos no es necesario en el ámbito de las redes de comunicaciones, y su proceso de búsqueda se extendería demasiado en el tiempo, de ahí que se haya limitado la difusión de etiquetas para acotar el número de caminos a descubrir.

La Figura 4.7 muestra el proceso de propagación de AMACs en el escenario de ejemplo de la Figura 4.6, dividiendo el proceso en tres imágenes que ocurren de forma simultánea: la propagación de dos etiquetas AMAC, y el proceso de aprendizaje de las mismas en los switches de la red. Además, en el ejemplo, Amaru está configurado con una profundidad de etiqueta de cinco niveles, tres AMACs aprendidas como máximo, y prefijo común menor de tres elementos. El proceso comienza en el controlador, que genera y envía el primer paquete de exploración con la etiqueta ARoot, representada en la imagen con un valor de *1*. Dicho paquete es recibido por el switch *S1*, que lo reenvía a través de todas sus interfaces excepto por la interfaz de entrada, añadiendo, por cada interfaz de salida, un nuevo elemento a la etiqueta AMAC del paquete. Este hecho da lugar a las Figuras 4.7a y 4.7b, que representan el proceso de propagación de etiquetas desde el switch *S1*. Mientras que la Figura 4.7a se centra en describir el proceso de difusión de la etiqueta *1.2* a través de la interfaz que conecta los switches *S1* y *S4*, la Figura 4.7b se encarga de representar la difusión de la etiqueta *1.3* a través de la interfaz que conecta los switches *S1* y *S2*. En cuanto al nuevo elemento añadido a la etiqueta AMAC, éste debe ser único para cada interfaz de un switch, siendo válido cualquier criterio de asignación que asegure dicho requisito (números generados aleatoriamente, número de puerto asociado a la interfaz, etc.). Finalmente, las tramas generadas en el switch *S1* son recibidas por los switches vecinos, que repiten la operación de reenvío, añadiendo a la etiqueta AMAC un nuevo elemento por cada interfaz de salida. El reenvío de las tramas sucede siempre que se cumplan las condiciones de P y L. Por ejemplo, en la Figura 4.7a, el mensaje con etiqueta *1.2.3.3.2* no es reenviado por *S7*, puesto que cuando lo recibe, la etiqueta ya cuenta con 5 niveles de profundidad. En la Figura 4.7b, el paquete enviado por *S4* hacia *S7* que incluye la etiqueta *1.3.2.1.2*, es rechazado por *S7* por compartir el prefijo *1.3.2* con la etiqueta *1.3.2.3.1* recibida previamente. Además, este paquete tampoco sería reenviado por *S7*, puesto que cuando lo recibe ya tiene una profundidad de 5 niveles. Una situación idéntica ocurre con el mensaje enviado de *S6* a *S9* con etiqueta *1.3.2.2.1*. Además, durante todo este proceso, los switches han aprendido las AMACs recibidas en los paquetes de exploración por orden de llegada, hasta alcanzar el cupo máximo de AMACs aprendidas en cada switch (N). Todos los switches tienen como máximo 3 AMACs aprendidas, rechazando el aprendizaje (pero no el reenvío) de todas las AMACs recibidas tras superar el cupo, como se puede ver en la Figura 4.7c, donde el switch *S9* rechaza la AMACs *1.2.2.1.1* por tener aprendidas previamente 3 AMACs. En relación al *S1*, éste no reenvía ninguna de las tramas difundidas por la red ni aprende ninguna AMAC, a excepción de la ARoot, dado que tiene una conexión directa con el controlador.

Una vez que el proceso de difusión y aprendizaje de AMACs termina, la red está lista



a) Propagación de la etiqueta 1.2.

b) Propagación de la etiqueta 1.3.

c) AMACs aprendidas en cada nodo.

Figura 4.7: Propagación de AMACs en el escenario de ejemplo.

para establecer la comunicación del plano de control entre el controlador y todos los dispositivos de la capa de infraestructura.

Aquí es preciso mencionar un trabajo que realiza una operativa similar al proceso de difusión y propagación de AMACs de Amaru para descubrir y mantener varios árboles de expansión desde un switch raíz [Acharya et al., 2020]. Sin embargo, su enfoque está más orientado a redes de switches tradicionales y su publicación es posterior a la publicación de Amaru.

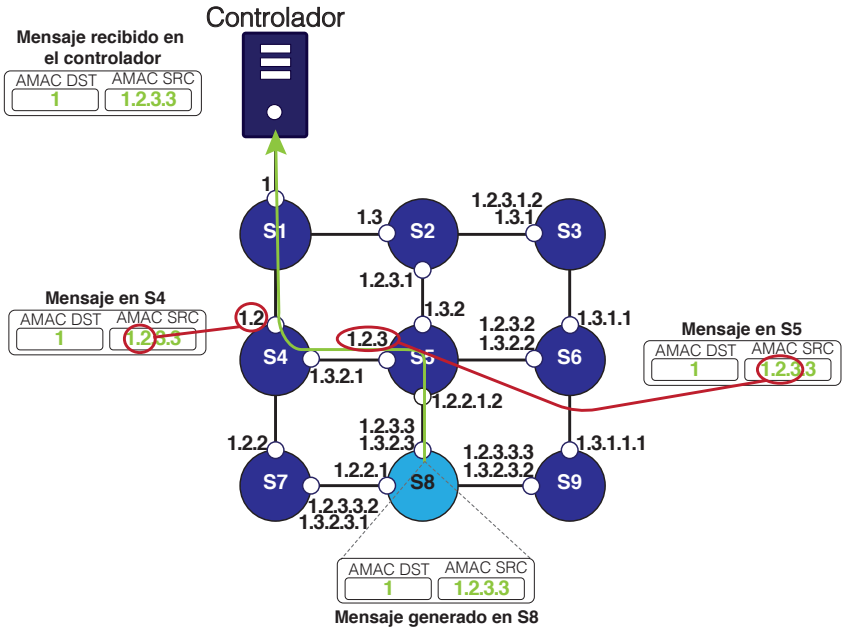
4.3 Lógica de comunicación y reenvío de paquetes entre controlador y switches

El proceso de construcción y aprendizaje de las AMACs proporciona un método de encaminamiento múltiple en cada uno de los switches de la red, que, a través de las AMACs aprendidas, son capaces de encaminar el tráfico de control hacia su destino. Cada switch, si quiere comunicarse con el controlador, tan solo tiene que escoger una de sus AMACs, incluirla como dirección origen del paquete de datos, e introducir la dirección del controlador (ARoot) como dirección destino. El controlador, para comunicarse con cada switch, tan solo tiene que escoger una de las AMACs asiganadas al switch en cuestión e introducirla como dirección destino del mensaje. Los switches intermedios que reciben los mensajes tan solo tienen que aplicar los principios de *source routing* ([Hamner & Samsen, 1988]) teniendo en cuenta las AMACs incluidas en el paquete y las AMAC aprendidas para reenviar los paquetes hacia su destino.

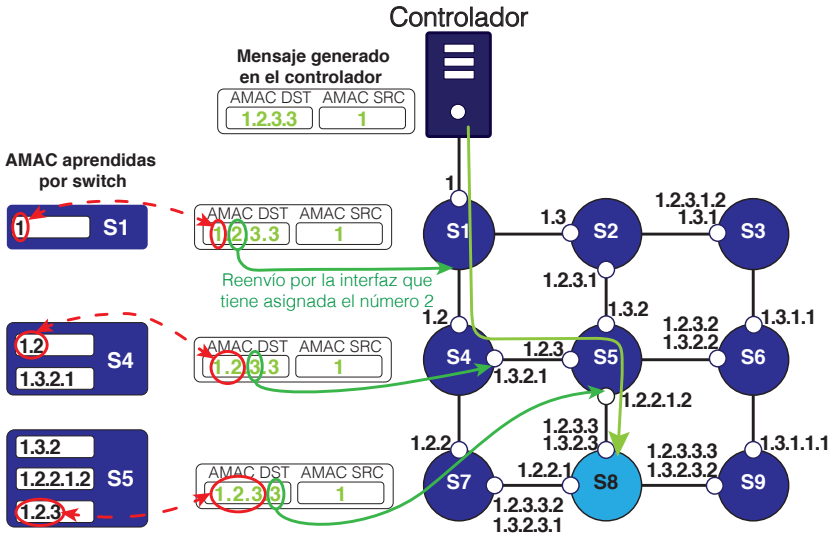
La Figura 4.8 muestra, sobre la topología de ejemplo, como se realiza el reenvío de los mensajes de control entre el controlador y los switches. En la Figura 4.8a, *S8* genera un nuevo mensaje con destino el controlador, por lo que en el campo del mensaje MAC destino incluye la AMAC controlador (ARoot), y en el campo de dirección MAC origen una de las AMACs aprendidas durante el proceso de difusión de las AMACs. En este ejemplo *S8* ha elegido la AMAC *1.2.3.3*, pero podría haber escogido cualquiera de las otras dos que tiene asignadas. Cuando *S5* recibe el paquete sabe que el destino del paquete es el controlador, por lo tanto, debe escoger el puerto adecuado para enviar el paquete hacia el controlador. Para ello, *S5* busca entre las AMACs aprendidas y selecciona aquella cuyo prefijo coincide con la AMAC origen del mensaje, para después reenviar el mensaje a través del puerto que tiene asociado. *S4* realiza el mismo proceso que *S5* para que el mensaje llegue a *S1* que, finalmente, entrega el mensaje al controlador.

El reenvío de mensajes en sentido controlador a switch se ilustra en la Figura 4.8b. Ahora el controlador genera un nuevo mensaje con destino a *S8*, por lo que incluye como

dirección MAC origen la ARoot y como dirección destino la AMAC *1.2.3.3*. Cuando el mensaje es recibido por *S1*, éste tiene que buscar la interfaz por la que reenviarlo, para lo que busca entre las AMAC aprendidas aquella que tenga una mayor coincidencia con la dirección MAC destino del paquete y, después, busca la interfaz cuyo identificador es igual al valor del elemento de la AMAC posterior a la coincidencia. En este ejemplo, el elemento posterior a la coincidencia es *2*, por lo que *S1* reenvía el mensaje a través de la interfaz cuyo identificador es *2*, la interfaz que comunica a *S1* con *S4*. Tanto *S4* como *S5* vuelven a aplicar el mismo mecanismo, buscar entre las AMAC aprendidas la que tenga mayor coincidencia con la dirección MAC destino del paquete, y reenviar el paquete por la



a) Envío de un mensaje desde S8 hacia el controlador.



b) Envío de un mensaje desde el controlador hacia S8.

Figura 4.8: Comunicación entre controlador y el switch S8.

interfaz cuyo identificador coincide con el elemento posterior a la coincidencia. Finalmente, el mensaje es recibido por *S8*.

No obstante, para que el controlador conozca las AMACs de los switches que gobierna, bien tiene que recibir previamente un mensaje de los switches y aprender la AMAC incluida en el paquete, bien ha de lanzar un proceso ARP [Plummer, 1982] o Neighbour Discover Protocol (NDP) [Narten et al., 2007] que descubra las la AMACs de dichos switches. En el caso de que el controlador ejecute un proceso ARP/NDP se ha de incluir algún mecanismo similar al de All-Path para garantizar que los mensajes llegan a todos los dispositivos sin provocar tormentas de mensajes.

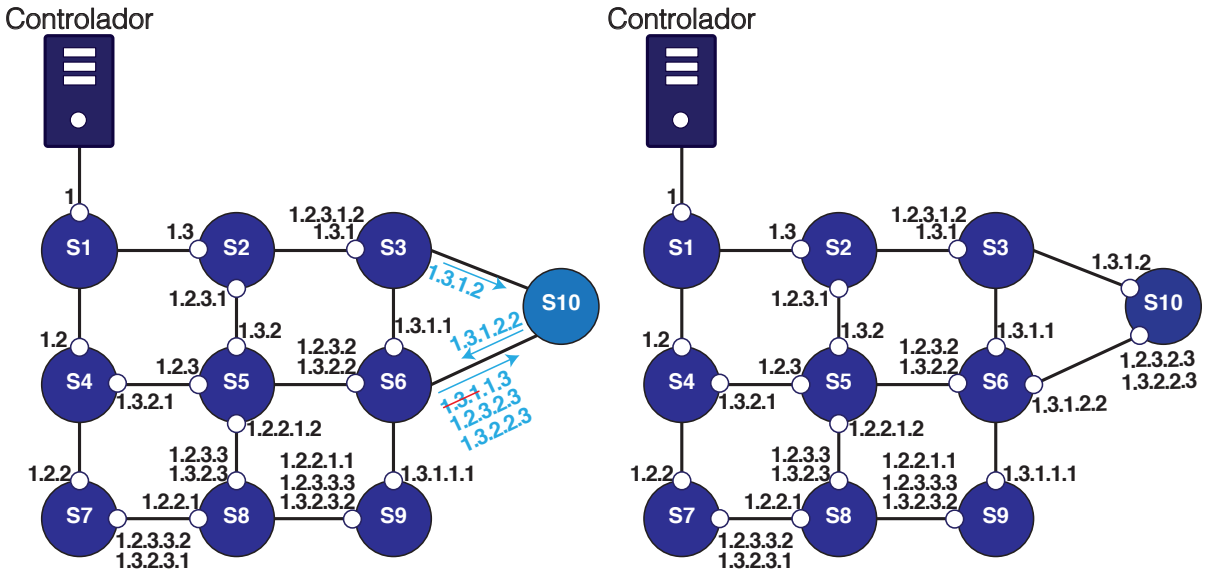
Gracias al sistema de sustitución de las direcciones MAC de los mensajes por etiquetas AMACs se garantiza que el tamaño de los paquetes permanece intacto, por lo que Amaru no necesita incluir nuevos campos de cabecera que sobrecarguen el paquete para comunicar el plano de control con el plano de datos.

4.4 Reconfiguración de red

Existen diversas situaciones ante las que es necesario ampliar o restablecer la configuración de encaminamiento del plano de control como, por ejemplo, al producirse caídas del servicio por fallos en el sistema, o al incorporar nuevos dispositivos. Estas situaciones pueden requerir la intervención manual del administrador de la red, siendo más habitual en el segundo caso. Sin embargo, Amaru elimina esta dependencia gracias a su proceso autoconfigurable que no requiere ninguna actuación por parte del gestor de la red. Únicamente necesita un mecanismo que le notifique la incorporación de nuevos dispositivos o la presencia de fallos en el sistema.

4.4.1 Reconfiguración por incorporación de nuevos miembros

En las redes de comunicaciones es habitual que los protocolos incorporen un mecanismo de "saludo" para dar a conocer la presencia de nuevos miembros en la red que, si se repite de forma periódica, sirve además como herramienta de verificación del estado de los dispositivos (si un dispositivo no envía o no responde a un "saludo" es que el dispositivo está en fallo). La primera descripción de este sistema de "saludo" data del año 1983 [Mills, 1983], y es incluido en protocolos tan conocidos como OSPF [Moy, 1998]. También es posible aprovechar el protocolo ARP para detectar la incorporación de nuevos miembros, ya que los switches SDN generan una petición ARP previa al establecimiento del canal de control para conocer la dirección IP del controlador. Amaru puede aprovechar estos mecanismos (o cualquier otro) para detectar la incorporación de nuevos switches y completar la configuración de encaminamiento del plano de control incluyendo al nuevo switch. Para comprender mejor el proceso se apoya la explicación en la Figura 4.9. Al escenario de ejemplo se ha incorporado el switch *S10*, que envía un



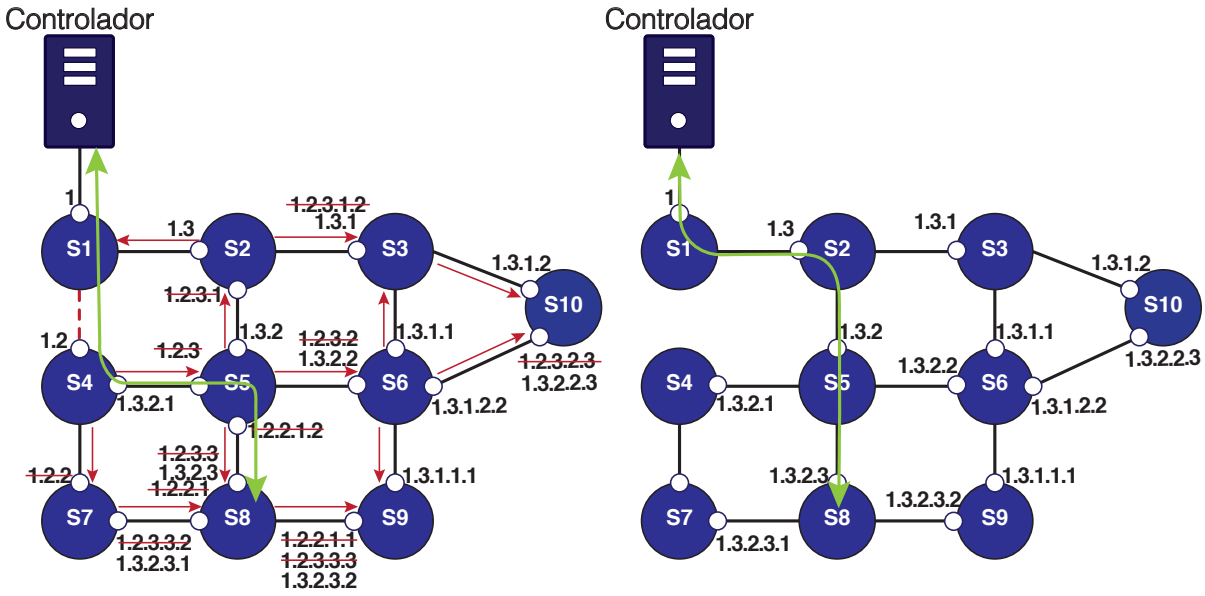
a) Propagación de nuevas AMACs con S10. b) Nuevas AMACs aprendidas.

Figura 4.9: Nuevo dispositivo incorporado en la topología de ejemplo.

mensaje de "saludo" a través de todas sus interfaces para darse a conocer. Este mensaje podría ser el paquete *ARP Request* que generan los switches al arrancar para conocer la dirección IP del controlador y poder establecer el canal de control, o algún mensaje de "saludo" similar a los que incorpora OSPF. Los switches vecinos (*S3* y *S6*), al recibir dicho mensaje, descubren que tienen un nuevo vecino, por lo que continúan con el proceso de expansión de AMACs. *S3* envía solamente la AMAC *1.3.1*, puesto que la AMAC *1.2.3.1.2* ha alcanzado la profundidad máxima de cinco saltos establecida en el escenario de ejemplo, mientras que el switch *S6* envía hacia el switch *S10* las tres AMACs que tiene aprendidas, de las cuales es rechazada la etiqueta *1.3.1.1.3* por compartir prefijo con la etiqueta *1.3.1.2* que llegó antes. El proceso concluye con el switch *S10* aprendiendo tres AMAC nuevas. Una vez que se han generado las nuevas rutas y el switch *S10* conoce la dirección IP del controlador se puede establecer el canal de control entre el switch *S10* y el controlador.

4.4.2 Reconfiguración por fallo de red

Que aparezcan fallos de red no es un evento frecuente, no obstante, si que se puede dar en casos puntuales, lo que ocasiona una interrupción del servicio que afecta a la experiencia del usuario. En cualquier red de comunicaciones el objetivo es que la experiencia de usuario sea impecable, por lo que la subsanación del fallo debe realizarse en el menor tiempo posible y, a poder ser, de forma automática, para que dicha reparación sea transparente e imperceptible por el usuario del servicio. Por suerte, Amaru con su sistema de generación, propagación y almacenamiento de AMACs, satisface esos requisitos, ya que es capaz de recuperar el sistema ante fallos de forma automática en un corto periodo de tiempo. El sistema de recuperación ante fallos de Amaru es muy sencillo y consiste en deshabilitar



a) Borrado del árbol con AMAC *1.2.X.Y.Z* por fallo de enlace.

b) Camino alternativo escogido por S8.

Figura 4.10: Borrado del árbol dependiente de la etiqueta *1.2* por fallo de enlace.

el árbol completo de la AMAC asociada a los enlaces o dispositivos afectados por el fallo, y seleccionar otra de las AMAC disponibles. La detección del fallo depende del mecanismo que tengan implementado los dispositivos de la capa de infraestructura, como por ejemplo un sistema que sondea periódicamente a los switches vecinos para detectar si alguno de ellos está en fallo. La Figura 4.10 representa el mecanismo de recuperación ante fallos de Amaru bajo la siguiente premisa: la comunicación del plano de control entre el controlador y el switch *S8* se corta debido a un fallo de red entre los switches *S1* y *S4*. El switch *S4*, tras detectar el fallo, notifica a sus vecinos que deben eliminar las etiquetas asociadas al prefijo *1.2* y éstos, a su vez, difunden la notificación a sus vecinos. La solicitud de borrado de etiquetas se expande por toda la red y, finalmente, todas las etiquetas aprendidas dependientes de la etiqueta *1.2* son eliminadas. El *S8* tras recibir la notificación de borrado elimina las etiquetas afectadas y, como la comunicación con el controlador dependía del árbol afectado por el fallo, selecciona una AMAC alternativa (*1.3.2.3*) para restablecer la comunicación. El controlador cuando recibe el primer mensaje con la nueva etiqueta actualizará la etiqueta asociada a *S8* y enviará los nuevos datos con la nueva AMAC como dirección destino. La restauración de la comunicación en el sentido switch-controlador es inmediata, ya que el switch conmuta automáticamente a otra AMAC ante un fallo, mientras que en el sentido contrario, el controlador tiene que esperar a recibir un mensaje con la nueva AMAC, por lo que el tiempo de recuperación se limita al tiempo que tarda el controlador en recibir el nuevo mensaje.

4.5 Características adicionales

La versatilidad que ofrece el proceso de asignación de etiquetas de Amaru permite trabajar con diferentes métricas de construcción de caminos entre el controlador y cada uno de los switches. Si se atiende al sentido estricto del proceso secuencial de construcción de las AMACs, éstas indican el número de saltos que tiene que atravesar un paquete para llegar hasta el controlador, lo que se traduce en que cuanto mayor es la longitud de la AMAC, mayor es la distancia al controlador, de ahí que apareciese el parámetro P para delimitar la profundidad máxima de las etiquetas y, por ende, el número máximo de saltos al controlador. Por lo tanto, el criterio que debería seguir un switch a la hora de elegir la AMAC principal, sería elegir la AMAC aprendida con menor profundidad. Sin embargo, el proceso de difusión de etiquetas basado en técnicas de exploración de red proporciona un ordenamiento alternativo de selección de AMAC basado en latencias. En este segundo ordenamiento tiene importancia el orden en el que se aprenden las AMACs, ya que la primera AMAC aprendida será la que proporcione el camino más rápido desde el controlador (independientemente del número de saltos atravesados). Es frecuente que los caminos con menor número de saltos aglutinen más tráfico debido, precisamente, a la elección del camino más corto como ruta por defecto, por lo que si un switch maneja una cantidad de tráfico mayor por una ruta, en dicha ruta aparecerán tiempos de espera en cola para los paquetes que demoran su llegada a destino. Por este motivo, un enfoque alternativo a las políticas de encaminamiento atendiendo al criterio de menor número de saltos es posible con Amaru.

Otra característica a resaltar de Amaru, es que puede ofrecer caminos disjuntos en enlaces entre el switch conectado directamente con el controlador (*S1* en los ejemplos vistos a lo largo de la sección) y el resto de switches que pertenecen a la red. Aquí es donde entra el juego el parámetro L, que especifica la longitud de prefijo que pueden compartir varias AMAC en un mismo switch, o lo que es lo mismo, los enlaces en común que pueden tener dos caminos. Si el parámetro L se fija en 2, se garantiza que los caminos son disjuntos, ya que dos AMAC en un mismo dispositivo solo podrán compartir un elemento (el enlace que va desde *S1* al controlador). Este parámetro y la posibilidad de obtener caminos disjuntos, ha sido la motivación de los siguientes capítulos de la tesis, centrados, precisamente, en la búsqueda de caminos disjuntos debido a las ventajas que ofrece esta propiedad.

Por otra parte, en el capítulo de revisión del estado del arte se ha visto que son varias las propuestas que permiten la convivencia simultánea de varios controladores SDN en la misma red SDN, o se comunican con otra red SDN a través de los dispositivos de la capa de infraestructura haciendo uso de un enfoque In-Band. Con Amaru ambos escenarios son posibles, tan solo se debe cumplir el requisito de que cada controlador tenga un identificador ARoot único. De esta forma, es posible tener varios procesos de propagación y asignación de etiquetas Amaru en paralelo garantizando, en todo momento,

la independencia de los procesos al no compartir el prefijo raíz ARoot. No obstante, en el caso de que existiese el solape de la etiqueta ARoot entre varios controladores, el proceso de propagación y aprendizaje de etiquetas continuaría, aunque aparecerían rutas superpuestas hacia ambos controladores que reducirían el número válido de caminos descubiertos, aunque la robustez del protocolo continuaría intacta.

4.6 Implementación de Amaru

La implementación de Amaru se ha llevado a cabo en tres plataformas distintas: una implementación en Python que simula el proceso de intercambio y aprendizaje de AMACs, una implementación en el simulador OMNeT++, y finalmente, en un switch SDN. El orden no es casual, sino que sigue un proceso lógico. En primer lugar, se ha modelado la propagación de las AMAC y su aprendizaje creando un simulador Amaru con un lenguaje de programación de alto nivel como es Python. El uso de Python como lenguaje de modelado se debe a la cantidad de librerías disponibles para el uso de los desarrolladores, así como por la simplicidad del lenguaje, lo que permite modelar sistemas en un corto periodo de tiempo. Esta primera aproximación permitió ver la viabilidad de Amaru, así como corregir errores de concepto de la versión inicial como, por ejemplo, la introducción de los parámetros P, N y L. En segundo lugar, se optó por utilizar un simulador de red basado en eventos discretos para evaluar el comportamiento de Amaru bajo un modelo de red más preciso y, por último, se utilizó un switch software SDN real como The basic OpenFlow userspace software switch (BOFUSS) [Fernandes et al., 2020] para demostrar que Amaru es viable en una red SDN real. En las tres implementaciones, para el proceso de construcción secuencial de etiquetas AMAC, se ha seguido el criterio de añadir a las etiquetas el número de puerto asignado a la interfaz de salida del switch, garantizando que el número asignado a cada interfaz es único. Además, las tres implementaciones están disponibles en el repositorio online [Lopez-Pajares et al., 2018].

4.6.1 Simulador Python

La primera aproximación consiste en un simulador construido expresamente para verificar el comportamiento de Amaru y depurar posibles errores de diseño. El simulador se centra en modelar el intercambio de mensajes entre los switches mediante un sistema basado en eventos de transmisión y recepción, por lo que este simulador no tiene en cuenta las latencias existentes en una red real (retardos de propagación, tiempos de transmisión y tiempo en cola). El evento de arranque comienza en el switch que está conectado directamente con el controlador, simulando la recepción del mensaje que contiene la etiqueta ARoot enviado desde el controlador. Este primer evento desencadena tantos eventos como vecinos tiene el switch inicial, lo que equivale al reenvío de paquetes a través de todas las interfaces del switch siguiendo la lógica de Amaru. Estos nuevos eventos

añaden un nuevo campo a la etiqueta AMAC que incluye el número de puerto asociado a la interfaz de salida del paquete, y programan un nuevo evento de recepción en el switch vecino. El evento de recepción comprueba en primer lugar si el nodo puede aceptar más AMACs (parámetro N) y, posteriormente, verifica que el prefijo recibido cumple las condiciones de longitud máxima (P) y longitud común de prefijo compartido con otras AMAC aprendidas (L). Si alguna de las condiciones no se cumple, se descarta el paquete, lo que no genera nuevos eventos. En caso contrario se vuelven a generar nuevos eventos para el reenvío del paquete a los switches vecinos.

4.6.2 Simulador de eventos discretos

Los simuladores de eventos discretos ofrecen modelos de red próximos a una red real, ya que en su estructura interna tienen en cuenta parámetros reales como son los tiempos de transmisión y propagación, y el tiempo de espera en cola, así como la definición de una pila de protocolos similar a la que implementan los sistemas reales. El instante de tiempo en el que se ejecuta cada evento lo marca un reloj interno que simula el concepto "tiempo" en la vida real. Por ejemplo, el evento de transmisión del paquete *"X"* ocurre cuando el paquete sale de la cola de transmisión (tiempo en cola). Al ejecutar el evento de transmisión, se fija el instante de tiempo en el que se sucede el evento de recepción, que viene determinado por la suma del tiempo de transmisión del paquete más el tiempo de propagación de dicho paquete por el enlace. De forma análoga, el simulador establece el tiempo de ejecución de todos los eventos que ocurren en una red de comunicaciones.

Dentro de los simuladores de eventos discretos, existen dos cuyo uso está bastante extendido en investigación: OMNeT++ [Varga, 2010] y ns-3 [Riley & Henderson, 2010]. Ambos se programan en C++ y tienen sus ventajas e inconvenientes. Sin embargo, para esta implementación se optó por OMNeT++, ya que es el simulador de eventos discretos utilizado por [Asadujjaman et al., 2018], lo que permitía replicar las pruebas de recuperación del sistema ante fallos.

En OMNeT++ se adaptó el sistema para trabajar con AMACs, y se incluyeron las restricciones de P, N y L para limitar las difusiones. Los procesos de transmisión y recepción de mensajes ya estaban implementados en el simulador, por lo que únicamente se hizo uso de la librería que los define. Además, el simulador incluye un sistema de notificación de enlace caído denominado *Link Failure Notification* por el que salta un evento en el caso de que un enlace falle. Este sistema de notificación es útil para llevar a cabo el proceso de borrado de las AMAC afectadas por un fallo de enlace, siguiendo los pasos de la Sección 4.4.2. Gracias al uso del simulador, además de los mensajes intercambiados, es posible obtener otras medidas, como el tiempo de convergencia del protocolo, o el tiempo de recuperación, magnitudes imposibles de cuantificar con el simulador en Python.

4.6.3 Switch software real

El último paso es implementar Amaru en un switch real. Se optó por utilizar un modelo de switch software en lugar de modificar un switch físico, ya que no se disponía de la financiación necesaria para montar un escenario de pruebas con una infraestructura de switches física. La ventaja de los modelos de switch software es que proporcionan un switch real que puede ser ejecutado en un ordenador convencional, que actúa como un switch. No obstante, como los ordenadores no tienen tantas tarjetas de red como interfaces tiene un switch, se necesita una plataforma de emulación que virtualice las interfaces del switch software. De hecho, con la plataforma de emulación, un mismo ordenador puede ejecutar varias instancias del switch software y conectarlas según demande la topología que se quiere montar. Esto es una ventaja, ya que con un único ordenador se pueden emular topologías que contienen varios switches. Además, el comportamiento de los entornos emulados se aproxima con bastante precisión a los sistemas reales, ya que se esta utilizando hardware físico para ejecutar el entorno de pruebas, lo que permite estudiar la viabilidad de Amaru en un entorno de pruebas similar a un escenario real.

En relación al switch software elegido, en la actualidad existen numerosos proyectos o soluciones que implementan switches software, sin embargo, en el ámbito de las redes SDN destacan dos propuestas, OVS [Pfaff et al., 2015], y BOFUSS [Fernandes et al., 2020]. Si bien OVS destaca por su rendimiento, su diseño interno es muy complejo, ya que utiliza funciones internas del kernel del sistema operativo para incrementar el número de paquetes que es capaz de procesar el switch. Por su parte, BOFUSS, cuenta con una estructura interna más sencilla a costa de reducir su rendimiento, lo que facilita y agiliza el prototipado de nuevos protocolos en BOFUSS. En la decisión final la balanza se decantó por BOFUSS, ya que es un switch perfectamente válido para evaluar Amaru que simplifica el proceso de implementación.

Para que BOFUSS soporte los procesos llevados a cabo por Amaru se ha modificado su arquitectura siguiendo el diagrama de flujo de la Figura 4.11. La estructura interna del switch BOFUSS se compone de un evento periódico ejecutado típicamente cada 100us que llama a la función *datapath*, encargada de comprobar si hay paquetes nuevos por procesar en cada una de las interfaces del switch. Si existe algún paquete nuevo, éste se envía a la función *pipeline* para que analice su contenido y actúe en consecuencia (reenvío del paquete hacia alguna interfaz, descarte del paquete, envío del paquete al controlador, etc.). Es en la función *pipeline* donde se introducen las modificaciones para que el switch trabaje con el protocolo Amaru (marcado en azul). En primer lugar, se comprueba si el paquete recibido corresponde a Amaru y, en caso negativo, se deja que el paquete sea procesado por el switch en su versión original. En caso de que sea un paquete Amaru, se comprueba el tipo de paquete y se toman las acciones oportunas para cada situación. A pesar de lo simple que puede parecer a priori la codificación de Amaru en BOFUSS, su integración en el switch requiere un trabajo de ingeniería inversa considerable para conocer

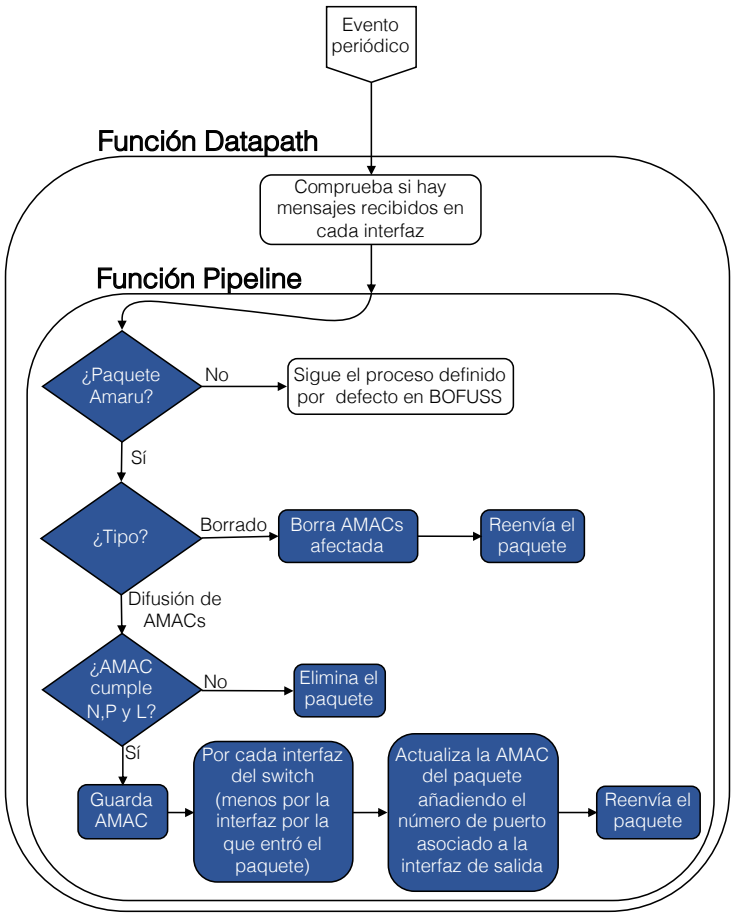


Figura 4.11: Diagrama de flujo de la implementación de Amaru en BOFUSS.

los entresijos del switch (sus estructuras de datos, las funciones que lo componen y los puntos de actuación). A todo esto se le suma que, aunque es un switch conceptualmente sencillo, carece de una documentación del código fuente como tal, por lo que hay que ir desgranando cada una de las funciones que lo componen. El tiempo invertido en el proceso de ingeniería inversa supera con creces el tiempo de codificación del protocolo. Este fue el principal motivo para ejecutar en último lugar la implementación de Amaru en un switch real, ya que era una pérdida de tiempo codificarlo en un sistema real sin antes haber comprobado que su funcionamiento era el correcto.

4.7 Evaluación de Amaru

La evaluación de Amaru llega después de definir, caracterizar y describir la implementación del protocolo en varias plataformas. Entre las características que se desean evaluar se encuentran la velocidad de autoconfiguración, la escalabilidad del protocolo y la fiabilidad ante caídas de la red, para lo que se va a medir el tiempo de convergencia de Amaru, el número de paquetes consumidos, y el ancho de banda mantenido durante la recuperación de fallos, que se corresponden con las tres características a evaluar, respectivamente.

El proceso de evaluación pretende, en primer lugar, validar las tres implementaciones de Amaru ejecutándolas sobre el mismo banco de pruebas. Posteriormente, una vez que se han validado las tres implementaciones, el objetivo es comparar Amaru con alguna propuesta alternativa. Sin embargo, ninguno de los trabajos analizados en el estado del arte se aproxima a Amaru, y de los que más se acercan, ninguno ofrece el código fuente para poder replicar los experimentos. Por ese motivo, se eligen los tres trabajos más cercanos a Amaru, ResilientFlow [Omizo et al., 2016], la propuestas de [Sakic et al., 2020], y [Asadujjaman et al., 2018], y se intenta hacer una comparativa justa. De ResilientFlow y [Sakic et al., 2020] interesa medir la velocidad de autoconfiguración y la escalabilidad, mientras que de la propuesta de [Asadujjaman et al., 2018] y otros interesan las cuestiones relativas al mecanismo de recuperación. ResilientFlow utiliza OSPF para encontrar las rutas entre el controlador y los switches de la red de manera autónoma, por lo que en el mejor de los casos, el tiempo invertido en establecer la comunicación vendrá determinado por el tiempo que tarda en converger OSPF, más el tiempo de configuración que necesite ResilientFlow, ocurriendo lo mismo con los paquetes utilizados para el intercambio de información. Por ese motivo, se van a utilizar como referencia el tiempo de convergencia y los paquetes generados por OSPF, ya que los valores obtenidos por ResilientFlow siempre serán mayores que los de OSPF. De forma análoga, [Sakic et al., 2020] hacen uso de RSTP para crear un árbol de expansión sobre el que aplica el proceso de configuración automático de la red, por lo que lo explicado para ResilientFlow se repite con el trabajo de [Sakic et al., 2020] y RSTP. Por lo tanto, en la comparativa del tiempo de convergencia y del número de paquetes utilizados por Amaru se van a utilizar los protocolos OSPF y RSTP. Con respecto al mecanismo de recuperación, se replica el escenario de pruebas de [Asadujjaman et al., 2018] para tener un valor de referencia.

Además, se utilizan tres modelos topológicos para evaluar el comportamiento de Amaru en diversos escenarios: una topología sintética regular (Figura 4.12), varias topologías sintéticas aleatorias, y una topología real. La topología sintética regular se usa como prueba de concepto para verificar que Amaru encuentra varios caminos entre el switch conectado al controlador y el resto de switches, mientras que las topologías aleatorias pretenden cubrir una variedad de situaciones y evaluar el comportamiento de Amaru en estructuras irregulares de tamaño variable. Por último, se utiliza la topología [Rogers Communications Inc., 2018] para evaluar la fiabilidad del protocolo, puesto que es la topología utilizada por [Asadujjaman et al., 2018] para medir los tiempos de recuperación y ancho de banda mantenido ante fallos de red.

Para generar las topologías de red aleatorias se utiliza el generador de topologías aleatorias Boston university Representative Topology gEnerator (BRITE) [Medina et al., 2001], que utiliza los modelos de Barabási-Albert [Barabási & Albert, 1999] y Waxman [Waxman, 1988] para generar las conexiones entre los nodos de la topología. Las topologías aleatorias utilizadas incrementan su tamaño de red (número de nodos) desde redes con 10 nodos, hasta redes de 60 nodos, en pasos de 10 nodos, y cada tamaño de red cuenta,

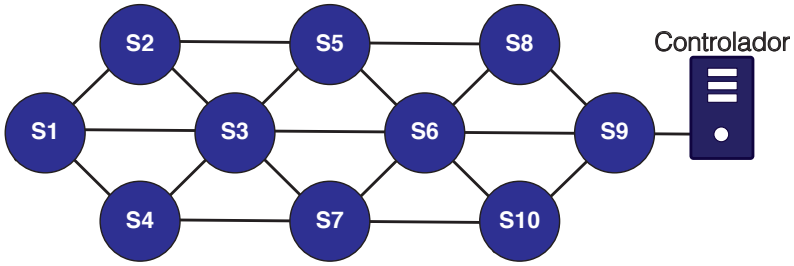


Figura 4.12: Topologia sintética regular.

a su vez, con varios grados de conectividad (3, 5 y 7 vecinos por cada nodo). Además, la distribución de los nodos en el espacio sigue una distribución Heavy-Tailed [Foss et al., 2011], emulando la distribución asimétrica de la población en el mundo real (grandes núcleos de población muy poblados con numerosas conexiones, y núcleos menores de población con menos conexiones). Con respecto al ancho de banda de los enlaces, éste se mantiene constante con una velocidad de transmisión de 1Gbps, mientras que el retardo de propagación del enlace depende de la distancia a la que se encuentren los nodos que une dicho enlace, menos en la topología regular, donde el tiempo de propagación de los enlaces se ha generado de forma aleatoria, dado que no existe una referencia de la distancia entre switches para calcular dicho tiempo. Además, cada experimento se repite 10 veces, con el objetivo de calcular intervalos de confianza.

A modo de resumen, la Tabla 4.1 recoge cada una de las pruebas realizadas y su justificación, para tener una visión global de todo el proceso de evaluación. No obstante, en la prueba del tiempo de convergencia se excluye a OSPF de la comparativa, debido al tiempo de convergencia obtenido por OSPF, que es 3 órdenes de magnitud superior al de Amaru. Al dibujar los resultados en una misma gráfica los resultados de OSPF enmascaran a los de Amaru y RSTP incluso representando el eje y en escala logarítmica, por lo que se han omitido.

Tabla 4.1: Resumen del proceso de evaluación de Amaru.

Prueba	Protocolo	Plataforma	Objetivo	Unidad	Razón
Validación	Amaru	Python OMNeT++ BOFUSS	Validar los modelos de Amaru	Número de paquetes	Modelado de Amaru y correción de errores
Tiempo de convergencia	Amaru RSTP	OMNeT++	Medir el tiempo que tardan los protocolos en obtener los caminos	Tiempo	Comprobar velocidad de autoconfiguración de Amaru
Paquetes consumidos	Amaru RSTP OSPF	OMNeT++	Medir la cantidad de paquetes que necesita cada protocolo	Número de paquetes	Comprobar que Amaru consume menos paquetes
Recuperación	Amaru Asadujjaman	OMNeT++ BOFUSS	Medir el tiempo de recuperación frente a fallos	Tiempo	Comprobar que Amaru recupera el camino en menos tiempo

4.7.1 Plataforma de validación

El banco de pruebas que ejecuta este apartado pretende dar por válidas las tres implementaciones de Amaru. En primer lugar, se ejecutaron dichas implementaciones sobre la topología de ejemplo, lo que sirvió para corregir errores sencillos. Sin embargo, estas pruebas no proporcionan un espacio muestral suficientemente representativo para encontrar errores de implementación que no son visibles a simple vista, por lo que se amplió la prueba a topologías aleatorias con un tamaño de red incremental de 10 a 60 nodos, manteniendo una conectividad media de tres vecinos por nodo. Además, se fijaron los parámetros de Amaru en P=100, N=8 y L=4 para fomentar la propagación de las AMACs sin limitar en exceso el número máximo de saltos (P=100), almacenando hasta ocho caminos alternativos en cada switch (N=8), y limitando a cuatro elementos el prefijo en común entre AMACs (L=4), lo que brinda un escenario perfecto para incentivar la multiplicidad de caminos sin que exista una confluencia de elementos comunes entre AMACs excesiva.

La Figura 4.13 recoge los resultados experimentales de este apartado, mostrando el número de paquetes consumidos por cada implementación en topologías aleatorias con modelos de conectividad Barabási-Albert y Waxman para distintos tamaños de red. En la gráfica se puede observar una ligera variación del número de paquetes consumidos entre las tres implementaciones en cada topología, siendo generalmente el simulador de Python es el que más paquetes consume. Esto se debe a que el simulador de Python simplemente se centra en modelar el proceso de intercambio de etiquetas, sin tener en cuenta otros

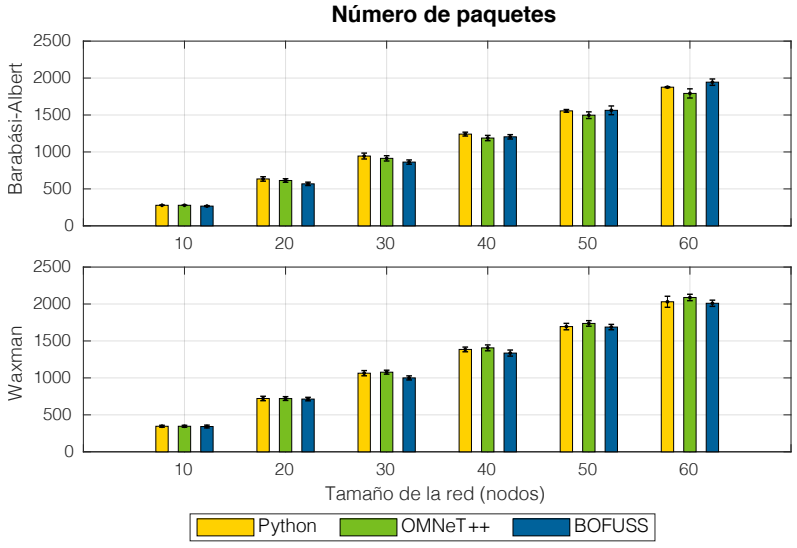


Figura 4.13: Número de paquetes consumidos por Amaru en sus tres implementaciones.

parámetros como los tiempos de propagación, transmisión y cola que si tienen en cuenta las otras dos plataformas, por lo que se procesan mas etiquetas AMAC. Algo similar ocurre entre el switch BOFUSS y el simulador OMNeT++, con el retardo de procesamiento de los paquetes, incluido el switch real, pero no en OMNeT++. No obstante, también existen discrepancias en el orden de salida de los paquetes por las interfaces de los switches al propagar la difusión de las AMAC. Mientras que en Python y OMNeT++ los paquetes salen por todas las interfaces en el mismo instante de tiempo, en BOFUSS el proceso es secuencial, por lo que existe un pequeño desfase temporal entre paquetes. Todo ello se traduce en que, en términos generales, el switch BOFUSS es el que menos paquetes consume, existiendo variaciones entre plataformas de un 5 % máximo, lo que valida las tres implementaciones para realizar pruebas más exhaustivas. No obstante, también es importante destacar otras medidas, como el número de caminos por switch. En todos los casos, independientemente de la plataforma utilizada, cada switch aprende ocho AMACs, cubriendo el cupo máximo establecido por el parámetro N, sin embargo, si que existen variaciones en la composición de dichas AMAC ya que no comparten los mismos prefijos en todas las plataformas, debido a la diferencia de retardos que introduce cada una de ellas.

4.7.2 Tiempo de convergencia

El segundo set de pruebas se centra en medir el tiempo de convergencia, definido como el tiempo que tardan los protocolos evaluados en establecer las rutas que conectan al controlador de la red con cada uno de los switches que componen la capa de infraestructura. El tiempo de convergencia proporciona un estimador fiable para valorar la velocidad de configuración de la redes SDN In-Band, cuyo valor depende en gran medida de tamaño de la red, ya que un mayor tamaño de red implica un mayor número de switches a evaluar, y un mayor tiempo de convergencia. No obstante, en Amaru el tiempo de convergencia depende también de los parámetros P, N y L, que son los responsables de controlar el aprendizaje y la difusión de las AMAC por la red, por lo que cuanto mayores sean dichos parámetros, más etiquetas se difundirán por la red durante un periodo mayor de tiempo. Además, a pesar de que en las redes SDN In-Band los datos del plano de control y del plano de datos comparten los mismos recursos físicos, las pruebas se han realizado sin tráfico de fondo, puesto que en las redes SDN para que circule tráfico por la red previamente se ha tenido que establecer la comunicación del plano de control.

La herramienta con la que se va a realizar esta comparativa es el simulador OMNeT++, ya que integra en dos librerías los protocolos RSTP y OSPF necesarios para realizar la evaluación del tiempo de convergencia. Como se ha explicado anteriormente, OSPF se ha excluido de la representación al obtener un tiempo de convergencia del rango de decenas de segundos que enmascara los resultados de Amaru y RSTP. Este tiempo de convergencia tan elevado se debe a que OSPF basa su diseño en el uso de temporizadores

para intercambiar información de encaminamiento entre vecinos, lo que alarga el proceso de intercambio de información. No obstante, para minimizar el tiempo de convergencia de RSTP y OSPF, se redujo al mínimo el valor de sus temporizadores (1s) para agilizar la operativa de ambos protocolos sin comprometer su estabilidad.

En Amaru, el tiempo de convergencia no depende de temporizadores, sino que depende de los parámetros P, N y L, por lo tanto, se van a variar dichos parámetros para evaluar el tiempo de convergencia de Amaru bajo dos supuestos, denominados mejor y peor caso. El mejor caso consiste en minimizar el tiempo de convergencia limitando la diversidad de caminos, ofreciendo una configuración de proporciona dos caminos por switch (N=2) y prefijo compartido de tres elementos (L=3), suficiente para tener un camino principal y otro de respaldo que no compartan muchos elementos del camino. El peor caso prima el descubrimiento de caminos a costa de penalizar el tiempo de convergencia, por lo que se aumenta el número de caminos aprendidos en cada switch hasta ocho (N=8), y se aumenta el prefijo hasta compartido hasta cuatro elementos (L=4). Además, para no añadir más complejidad a la prueba, no se ha limitado la profundidad de la etiqueta.

La Figura 4.14 recoge el resultado del tiempo de convergencia obtenido con Amaru y RSTP en los dos modelos de topologías aleatorias con distinto grado de conectividad entre vecinos. La fila superior muestra los resultados en las topologías con un modelo Barabási-Albert, mientras que la fila inferior recopila los resultados para Waxman. Con respecto a la distribución vertical de la figura, las gráficas situadas en la parte izquierda muestran los resultados en topologías con una conectividad media de tres vecinos por switch, las centrales para topologías con cinco vecinos por switch y las de la izquierda con siete vecinos por switch. Además, el color amarillo representa el tiempo de convergencia de Amaru en el mejor caso (menor diversidad de caminos), el color verde en el peor caso, y el color azul el tiempo de convergencia de RSTP.

Independientemente de juego de resultados, el tiempo de convergencia se incrementa conforme aumenta el tamaño de la red, ya que se ha de difundir y procesar una mayor cantidad de información por la red. Sin embargo, los resultados apenas varían si se compara el mismo protocolo en el mismo modelo topológico con el mismo tamaño de red, pero distinto grado de conectividad. Esto se debe a que un incremento en el grado de conectividad implica incrementar el ratio número de enlaces por switch de la topología, lo que evidentemente incrementa el número de caminos disponibles y eleva ligeramente el tiempo de convergencia. Sin embargo, no se incrementa el número de switches, por lo que no hay que procesar o visitar nuevos switches e intercambiar su información, de ahí que no aumente significativamente el tiempo de convergencia. La evidencia de este razonamiento es que un incremento del número de enlaces tiene un efecto nulo en RSTP, ya que este protocolo descubre un único árbol de expansión que contiene un único camino desde cada switch hasta el switch raíz, por lo que un aumento del número de enlaces no afecta al tiempo de convergencia. En Amaru, sus políticas de reenvío limitan la cantidad de mensajes reenviados al número máximo de AMACs por switch, por lo que cuando se

alcanza el cupo de AMACs aprendidas no se propagan más etiquetas independientemente del número de enlaces de la topología.

Con respecto a la diferencia de tiempos obtenida por Amaru y RSTP, Amaru resulta vencedor al reducir el tiempo de convergencia en tres órdenes de magnitud. Por lo tanto, Amaru es más rápido que la propuesta de [Sakic et al., 2020] gracias al proceso de difusión de etiquetas que elimina el uso de temporizadores y tiempos de espera muertos. Con respecto a ResilentFlow y OSPF, Amaru consigue reducir el tiempo de convergencia cinco órdenes de magnitud. Por lo tanto, Amaru se sitúa muy por encima de sus competidores en relación al tiempo invertido en obtener los caminos para establecer el canal de control. Además se recuerda que cada experimento se ha ejecutado 10 veces y se han calculado los intervalos de confianza al 95 %, mostrando que no existen grandes variaciones entre las ejecuciones.

4.7.3 Paquetes consumidos

El segundo set de pruebas se centra en medir el número de paquetes consumidos por Amaru, OSPF y RSTP, permitiendo evaluar la eficiencia de dichos protocolos en relación a los recursos de red consumidos, y estimar así la escalabilidad de las propuestas. Para este conjunto de pruebas se ha vuelto a utilizar el simulador OMNeT++, y se han incluido los resultados de OSPF en las gráficas, obteniendo una comparativa triple con Amaru, RSTP

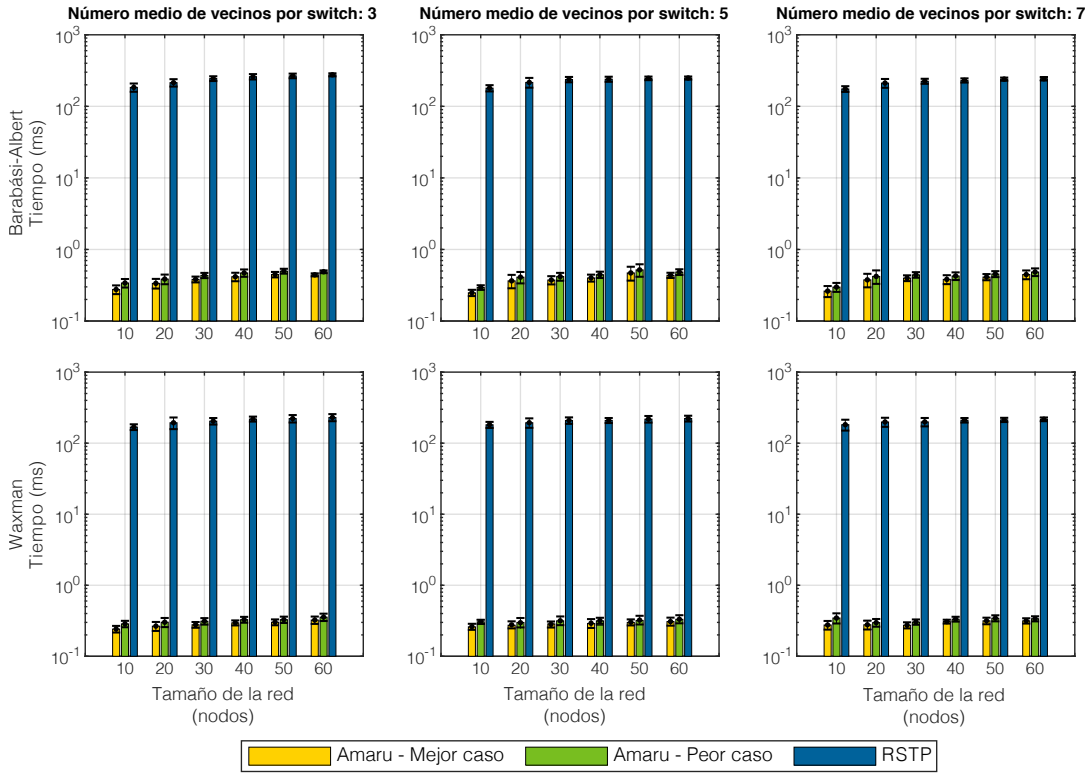


Figura 4.14: Tiempo de convergencia obtenido por Amaru y RSTP.

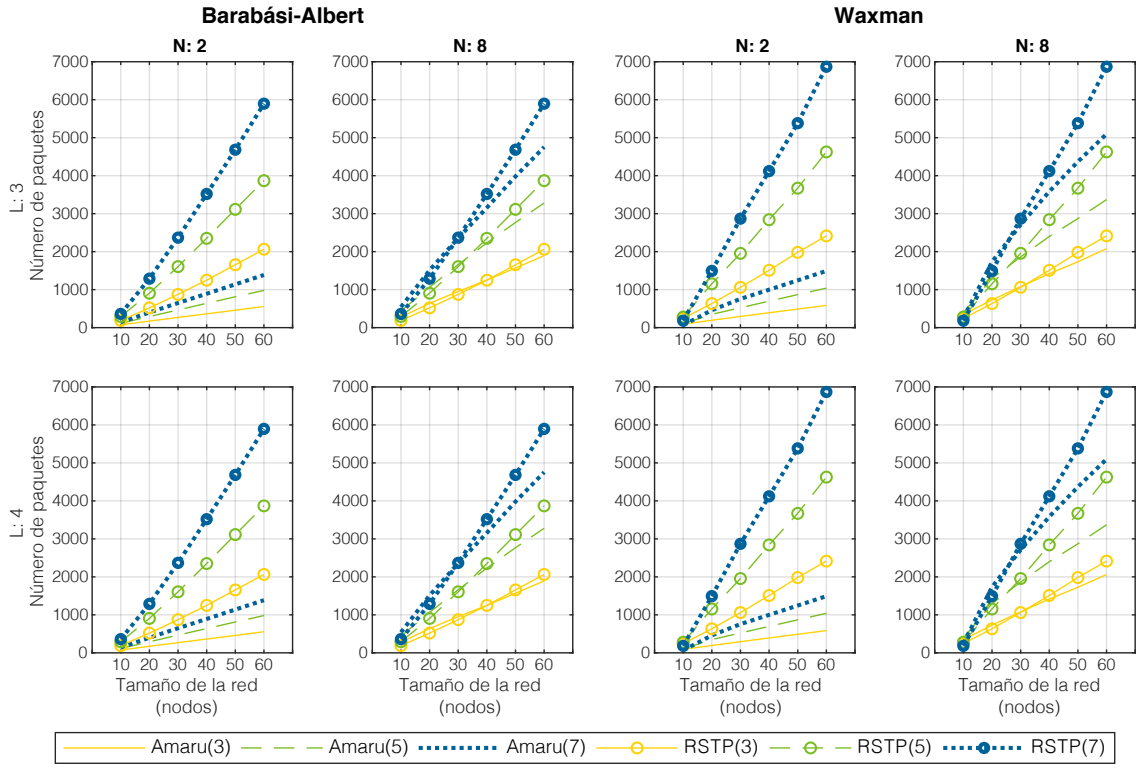


Figura 4.15: Paquetes consumidos por Amaru y RSTP.

y OSPF. Al igual que en el apartado anterior, se van a ejecutar instancias de Amaru con distintos parámetros de configuración (N={2, 8} y L={3, 4}), y se va a realizar una comparación a pares, entre Amaru y RSTP, y Amaru y OSPF. La Figura 4.15 muestra el número de paquetes consumidos por Amaru y RSTP sobre el mismo juego de topologías que la sección anterior.

La Figura 4.15 contiene la comparativa de paquetes entre Amaru y RSTP, cuya estructura se divide en dos bloques de cuatro gráficas cada uno. Las cuatro gráficas de la mitad izquierda se corresponden con los resultados en las topologías con modelo de conectividad Barabási-Albert, y las de la derecha con el modelo de Waxman. En cuanto a la división por filas, la fila superior recoge los resultados para un prefijo común de tres elementos (L=3) y la inferior para un prefijo de cuatro elementos (L=4), mientras que la división por columnas recopilan los resultados para valores de N=2 y N=8. Además, cada gráfico contiene tanto los paquetes consumidos por Amaru, como los consumidos por RSTP en los tres grados de conectividad entre vecinos (3, 5 y 7). El color amarillo con línea continua se utiliza para las topologías con un grado de conectividad tres, el color verde con línea discontinua para las de grado de conectividad cinco, y el color azul con línea punteada para un grado de conectividad siete. Además, los resultados de RSTP incluyen un marcador en forma de círculo para diferenciarse de los resultados de Amaru. A grandes rasgos, RSTP necesita un 7,7 % más de paquetes que Amaru para establecer las rutas entre el controlador y los switches de la red (valores medios recogidos entre todo el

juego de resultados). Sin embargo, se aprecia como la diferencia de paquetes consumidos entre Amaru y RSTP se incrementa conforme aumenta el tamaño de la topología, dónde el número de paquetes consumidos por RSTP aumenta su distancia con respecto al número de paquetes utilizados por Amaru, independientemente del modelo de conexión de la topología, o de la conectividad entre switches de la misma, incluso en el escenario del peor caso para Amaru (N=8 y L=4). No obstante, también se observa que la diferencia entre los resultados de Amaru para prefijos L=3 o L=4 no es significativa, a diferencia de lo que ocurre con el parámetro N, en que, para valores de N=8, el número de mensajes utilizados por Amaru llega a cuadruplicarse con respecto a N=2, debido, principalmente, a que con N=8 se obtienen 4 veces más de direcciones por switch que con N=2, lo que incrementa la difusión de etiquetas por la red y el por ende, el número de paquetes. Aún en el peor de los casos, los valores obtenidos por Amaru siguen siendo mejores que los obtenidos por RSTP.

La comparativa entre Amaru y OSPF sigue la misma estructura que la realizada entre Amaru y RSTP, quedando recogidos los resultados de esta nueva comparativa en la Figura 4.16. En esta nueva ilustración cambia el eje vertical de las gráficas, representándolo en unidades logarítmicas para garantizar la visualización de los datos, debido al elevado número de paquetes que necesita OSPF. Aquí, la diferencia de paquetes utilizados es aún mayor que en el caso anterior, ya que OSPF multiplica como mínimo por 10 el numero de paquetes utilizados con respecto a Amaru. En media necesita 30 veces más

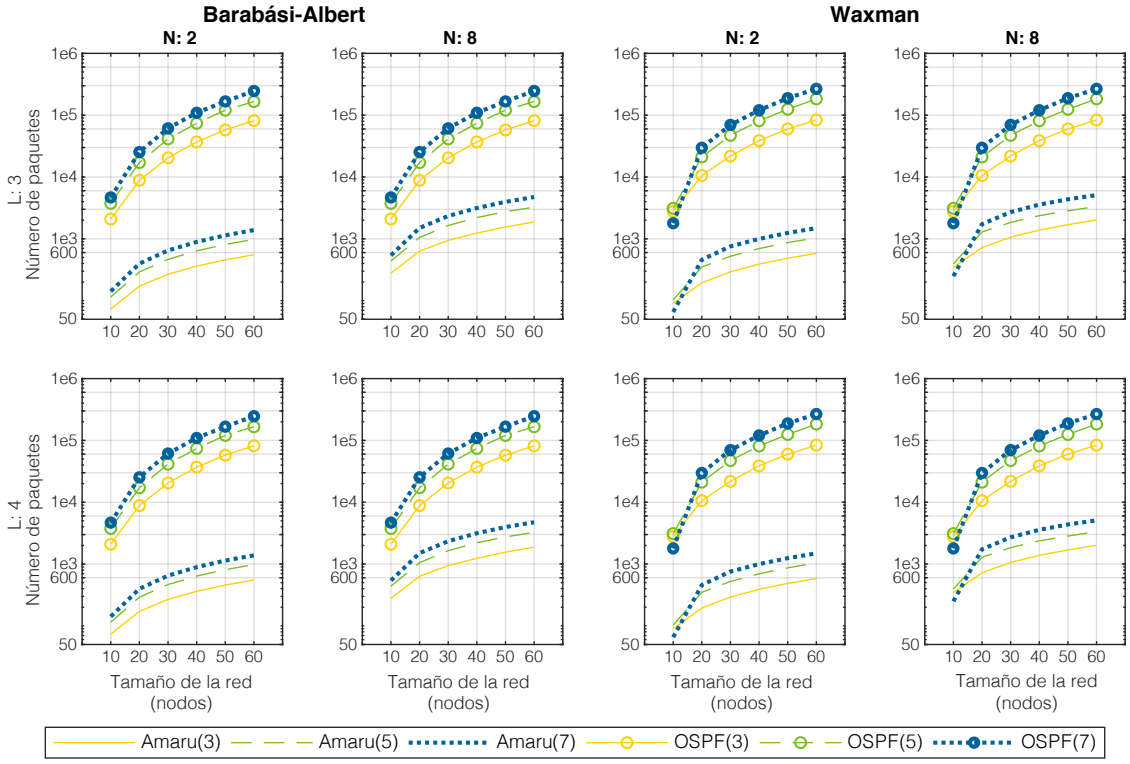


Figura 4.16: Paquetes consumidos por Amaru y OSPF.

debido, principalmente, a que en OSPF cada nodo actualiza la información de sus vecinos cada vez que ocurre una actualización del estado de la red, mientras que Amaru genera información solamente durante la difusión de etiquetas. Este también es el motivo por el que OSPF obtiene unos tiempos de convergencia muy superiores a los recogidos por Amaru.

A la vista de los resultados, Amaru necesita generar muchos menos paquetes que sus competidores, ofreciendo una alternativa escalable para el establecimiento de múltiples caminos entre controlador y switches en redes SDN In-Band. Si bien la diferencia con RSTP en redes de tamaño pequeño no es grande, Amaru aventaja a RSTP al ofrecer múltiples caminos entre cada uno de los switches de la red y el controlador frente al camino único que ofrece RSTP desde cada switch al controlador. En el caso de OSPF, esta propuesta si que obtiene múltiples caminos entre los switches de la red y el controlador, pero a costa de multiplicar por 30 el número de paquetes utilizados con respecto a Amaru. Por lo tanto, Amaru se presenta como una alternativa escalable frente a los trabajos de [Sakic et al., 2020] (RSTP) y ResilentFlow (OSPF) para generar caminos de forma autónoma entre el controlador y los switches de la red en redes SDN In-Band.

4.7.4 Tiempo de recuperación

El último apartado de la evaluación se centra en valorar el sistema de recuperación ante fallos de red de Amaru midiendo el tiempo de recuperación y el ancho de banda mantenido durante la recuperación de uno o varios fallos de red. Además, en este último test, se utilizan dos implementaciones de Amaru sobre dos entornos de pruebas distintos. El primer entorno utiliza el switch BOFUSS y dos topologías: la topología sintética regular y la topología Rogers. Se elige la topología sintética regular porque es una topología sencilla sobre la que ejecutar BOFUSS, lo que permite estimar manualmente el tiempo de recuperación y comprobar que los datos experimentales se aproximan a la estimación teórica. Después, se utiliza la topología Rogers para ver como se comporta Amaru con fallos en cadena sobre una topología real. El segundo entorno replica con Amaru el escenario de pruebas de [Asadujjaman et al., 2018] utilizando el simulador OMNeT++, lo que proporciona un valor de referencia sobre la calidad de respuesta de Amaru a la restauración de fallos. Para construir las topologías e integrar los switches BOFUSS en ellas, se utiliza Mininet [Kaur et al., 2014], un framework que emula redes de comunicaciones integrando código software y funciones del kernel de sistemas Linux. Gracias a Mininet se puede emular dentro de un ordenador una red de comunicaciones completa, no obstante, su escalabilidad está limitada, por lo que si se quieren emular redes de gran tamaño hay que ampliar el número de ordenadores y comunicarlos entre sí.

La Figura 4.17 recoge el primer escenario de la primera prueba, un ambiente con doble fallo en la topología sintética regular. En dicho escenario Amaru previamente ha descubierto tres caminos, siendo el camino principal, coloreado en verde, el que mantiene

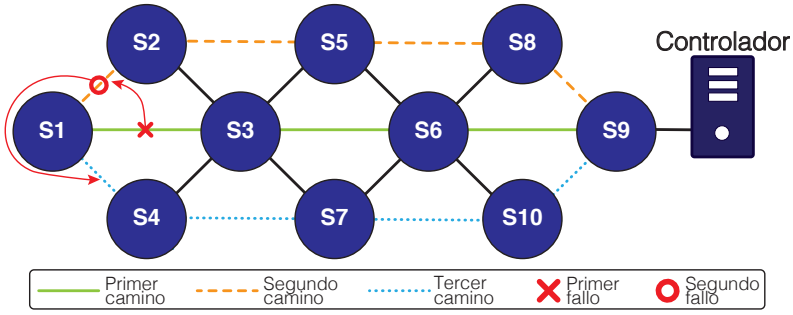


Figura 4.17: Escenario con doble fallo en topología sintética regular.

activa la comunicación del plano de control entre el switch *S1* y el controlador. Al segundo de iniciarse la comunicación ocurre un fallo en el enlaces entre los switches *S1* y *S3*, por lo que el switch *S1* conmuta automáticamente al segundo camino (color naranja). En el segundo dos vuelve a ocurrir un segundo fallo entre los switches *S1* y *S2*, por lo que el switch *S1* vuelve a conmutar automáticamente al tercer camino que tiene aprendido. Además, con el objetivo de medir el tiempo de recuperación en ambos sentidos de la comunicación, se realizan las medidas de ancho de banda y tiempo de recuperación en sentidos contrapuestos, de controlador a switch y de switch a controlador. Para calcular el tiempo de recuperación se van a lanzar dos flujos de datos con velocidad constante entre *S1* y el controlador en sentidos opuestos, y se va a medir el tiempo entre llegadas de sus paquetes, o Inter Arrival Time (IAT), en ambos sentidos. De esta forma, se estima el tiempo de recuperación, ya que el IAT permanecerá constante cuando la red está estabilizada, y se incrementará cuando se produzcan fallos en la red como consecuencia del tiempo que se tarda en restaurar la ruta. Para garantizar un flujo de comunicación constante se utiliza el protocolo UDP, ya que este protocolo no introduce mecanismos de reenvío o de control de la congestión que hacen fluctuar el IAT. Además, el flujo cuenta con una tasa de transmisión constante de 750Mbps, suficiente para cargar los enlaces de la red al 75 %, por lo que no se llegan a saturar. Con estos datos, junto con un tamaño de paquete de 1500 *bytes* (el tamaño máximo de una trama Ethernet), se obtiene un IAT de 16μs en media, lo que garantiza una granuralidad suficiente para medir la recuperación del sistema, ya que los tiempos de recuperación se sitúan al menos en un orden de magnitud superior. Por lo tanto, el tiempo de recuperación va implícito en el IAT del primer paquete recibido tras sufrir el fallo, ya que ese paquete incluye el tiempo de recuperación del sistema.

Los resultados del primer escenario de la primera prueba (doble fallo en la topología sintética), los recopila la Figura 4.18, cuya mitad izquierda muestra los resultados vistos de switch a controlador, y la parte de la derecha los resultados vistos de controlador a switch. Además, la fila superior recoge el IAT, y la fila inferior el ancho de banda mantenido durante el transcurso de la prueba. El IAT del flujo en sentido switch a controlador se mide en el controlador, mientras que el flujo en sentido contrario se mide en el switch. Con respecto al ancho de banda, éste siempre se mide en el switch para ver como afecta el cambio de puerto en el switch *S1* (la numeración de los puertos se corresponde con el

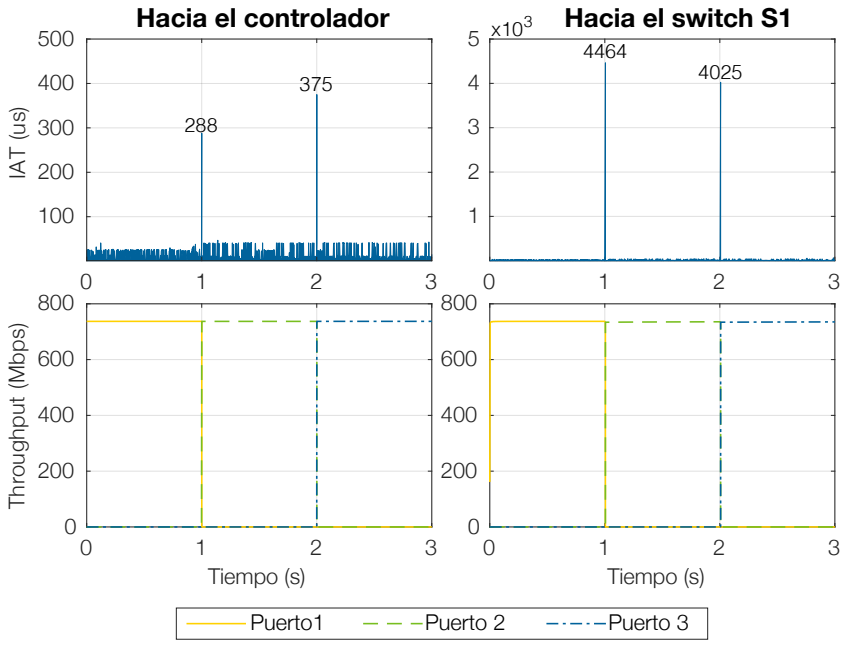


Figura 4.18: Escenario con doble fallo en topología sintética regular.

orden de los caminos). El IAT sufre un incremento notable en los instantes de tiempo 1s y 2s, momentos en que se producen los fallos de red. Sin embargo, se observa que el tiempo de recuperación en el sentido switch a controlador es menor que en sentido inverso. Esto se debe a que el switch conmuta directamente a la ruta de respaldo al detectar el fallo, por lo que el tiempo de recuperación es el tiempo que tarda en conmutar el switch, más la diferencia de tiempos entre el camino principal y el de respaldo. En sentido inverso el tiempo de recuperación es mayor, porque el controlador no es consciente del fallo hasta que le llega la nueva AMAC del switch *S1* por el camino de respaldo, que es el tiempo adicional que aparece en la gráfica con respecto al sentido switch controlador. Además, en la gráfica del ancho de banda se observa que la conmutación entre caminos es continua y el ancho de banda no se ve afectado por el fallo. En el peor de los casos, el tiempo de recuperación asciende a 4ms, situándose un orden de magnitud por debajo de los 50ms que fijaron [Niven-Jenkins et al., 2009] como tiempo máximo de recuperación.

El segundo escenario de la primera prueba traslada el experimento a la topología Rogers y aumenta el número de fallos consecutivos hasta 4. La metodología y las métricas evaluadas son las mismas que en el escenario anterior. Con estos cuatro fallos consecutivos se evalúa el sistema de recuperación de Amaru en puntos dispares de la red, viendo si Amaru es capaz de recuperar el sistema en menos de 50ms, tanto en puntos cercanos al controlador como en puntos lejanos. La Figura 4.19 muestra el escenario de la prueba, donde se puede ver la topología Rogers junto con los retardos de propagación de cada uno de los enlaces, y los puntos de fallo. El controlador se encuentra ubicado en el nodo Toronto, y existe una comunicación activa entre el controlador y el switch ubicado en Victoria a través del camino coloreado en verde.

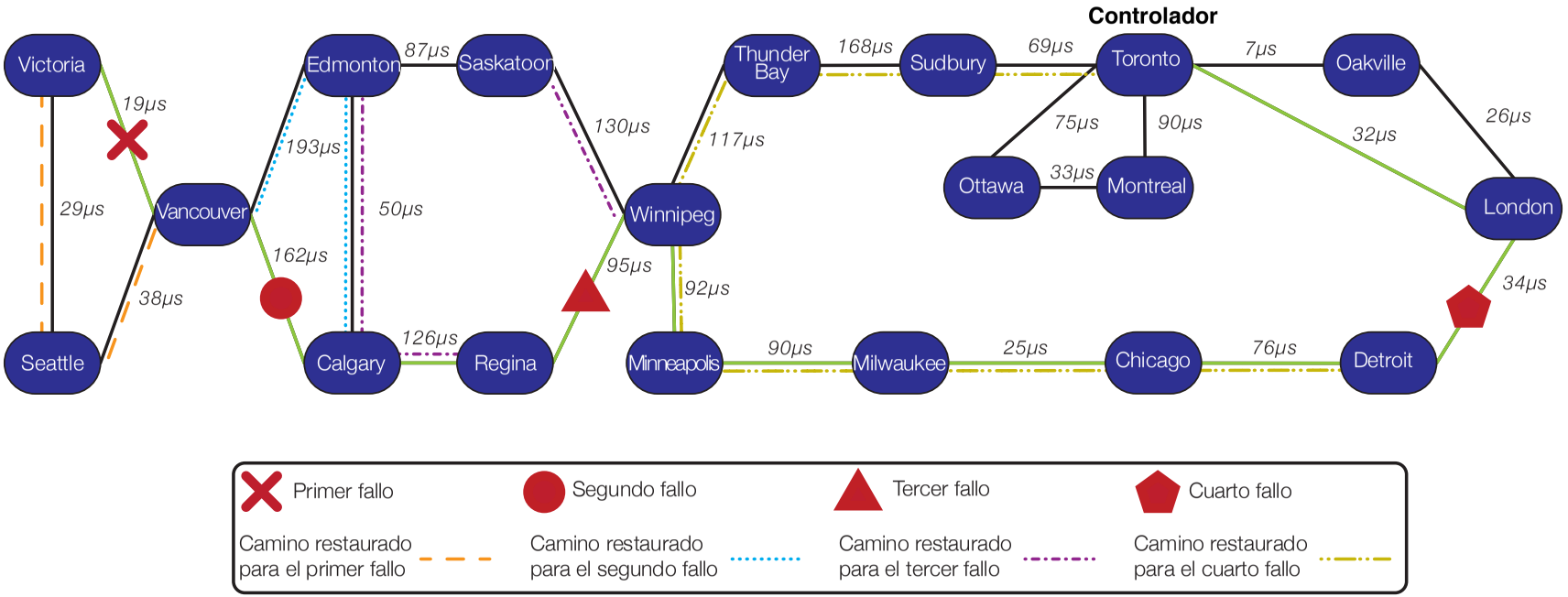


Figura 4.19: Topología Rogers con retardos de enlace.

Durante el transcurso de la prueba ocurren cuatro fallos consecutivos, el primero entre los switches Victoria y Vancouver, el segundo entre Vancouver y Calgary, el tercero entre Regina y Winnipeg, y el cuarto entre Detroit y London. Merecen una atención especial el segundo y el cuarto fallo, ya que la recuperación del fallo incluye un retroceso en la ruta para encontrar un camino alternativo. No obstante, solamente retrocederán aquellos paquetes que están en tránsito por la rama afectada durante el proceso de restauración, ya que una vez que la ruta es restaurada en los switches Calgary o Winnipeg, el tráfico se reenvía directamente al controlador por la nueva ruta. Este efecto provoca un desorden de paquetes en el controlador, sin embargo, los protocolos de la interfaz southbound como OpenFlow resuelven este problema al utilizar TCP, que resuelve el problema de la ordenación y la pérdida de paquetes.

La Figura 4.20 recoge los resultados de esta prueba, cuya estructura es igual que la de la Figura 4.18, solo que ahora la fila inferior de la figura muestra el ancho de banda del flujo medido en el controlador y en el switch, en lugar de mostrar solamente el ancho de banda en las interfaces del switch. El análisis de los resultados lleva a la misma conclusión que la prueba anterior, el sentido de comunicación desde controlador a switch necesita más tiempo en recuperar la conexión que el sentido inverso debido al proceso de notificación del fallo en el controlador. Sin embargo, se observa que los tiempos de recuperación son ligeramente superiores que la diferencia de tiempos entre el camino principal y el de respaldo debido, precisamente, a que el proceso de conmutación al camino de respaldo en un switch software real es rápido, pero no inmediato, ocurriendo lo mismo con el tiempo de procesamiento de los paquetes. Estos tiempos, que no se tienen en cuenta en el cálculo teórico, son los que incrementan el tiempo de recuperación. A pesar de estos retardos

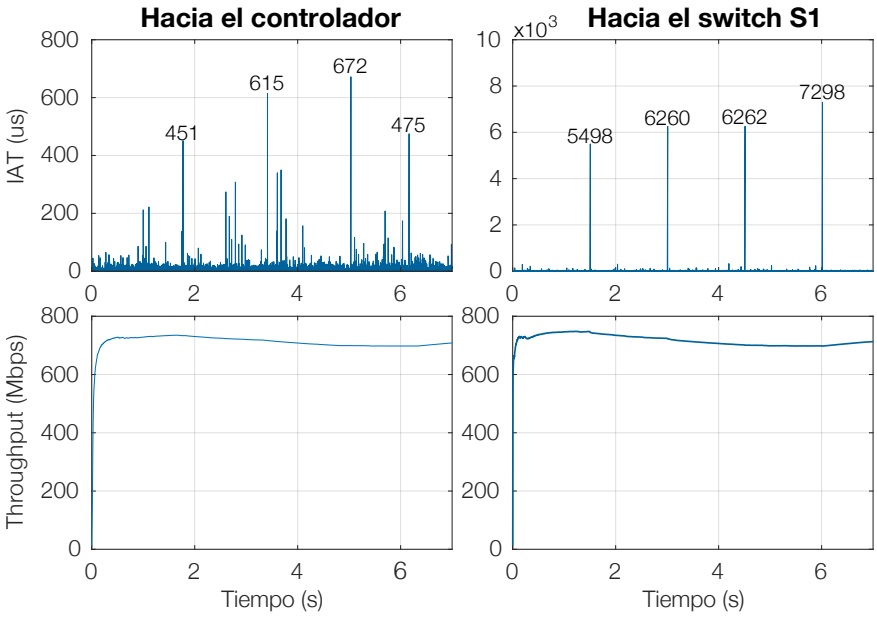
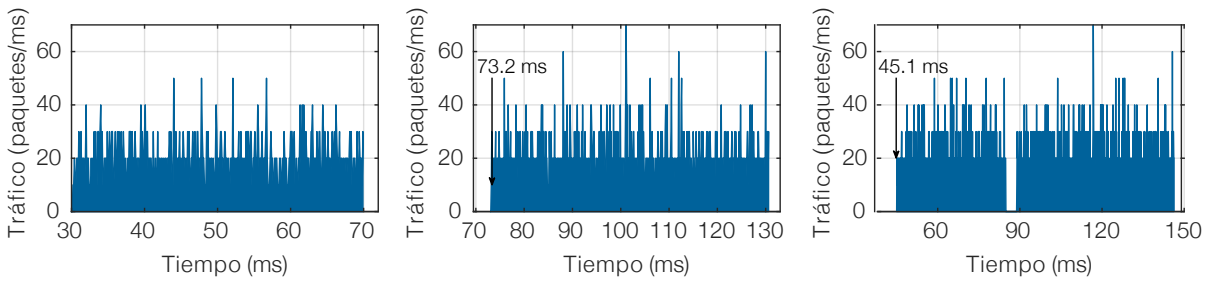


Figura 4.20: Escenario con doble fallo en topología Rogers.

adicionales, en el peor de los casos, se restaura la ruta en unos 7ms, por lo que el tiempo de recuperación se sigue situando muy por debajo de los 50ms. Además, gracias a la rápida restauración de las rutas el ancho de banda apenas se ve mermado.

El segundo entorno replica con Amaru el escenario de pruebas de [Asadujjaman et al., 2018] para tener una comparación objetiva de la calidad de Amaru recuperando fallos de red. Las pruebas se realizan con el simulador OMNeT++ y la topología Rogers, produciéndose un fallo de red en el enlace que une los nodos Victoria y Vancouver. Esta prueba cambia el sistema de medición con respecto a las medidas realizadas en los dos escenarios de la prueba anterior. Ahora se mide el número de paquetes por segundo que transitan por tres enlaces, el enlace afectado por el fallo entre los nodos Victoria y Vancouver, el enlace del camino de backup entre Victoria y Seattle, y el enlace que conecta al switch Toronto con el controlador. El tiempo de recuperación viene ahora determinado por el periodo de tiempo en el que el controlador deja de recibir paquetes, u obteniendo la diferencia de tiempos entre el momento en el que el enlace que presenta fallo deja de reenviar tráfico, y el instante de tiempo en el que el enlace recuperado empieza a recibir dicho tráfico. Sin embargo, este sistema de medición no tiene en cuenta como afecta la recuperación en ambos sentidos de la comunicación. La prueba comienza transmitiendo un flujo de datos desde el switch Victoria hasta el controlador en el instante de tiempo t=30ms, produciéndose un fallo en el enlace en el instante t=70ms.

La Figura 4.21 recoge los resultados de la prueba, observándose en la Figura 4.21a el inicio de la comunicación a los 30ms y el fallo del enlace a los 70ms. La Figura 4.21b muestra como el enlace recuperado empieza a recibir paquetes en el instante t=73,2ms, por lo que el tiempo de recuperación es de 3.2ms. Este tiempo de recuperación se corresponde también con la franja en blanco de la Figura 4.21c, donde el controlador no recibe paquetes. Sin embargo, se observa como esta ausencia de paquetes no ocurre en las proximidades del instante t=70ms, que es donde se produce el fallo, sino que se retrasa unos 15ms. Este efecto se debe a que transcurren 15.1ms desde que se envían los paquetes del switch Victoria hasta que llegan al controlador, como se puede apreciar al observar las Figuras 4.21a y 4.21c y restar el instante de tiempo del primer paquete de ambas gráficas. Los resultados obtenidos por [Asadujjaman et al., 2018] en la misma prueba arrojan un



a) Enlace sometido a fallo. b) Enlace recuperado. c) Controlador.

Figura 4.21: Resultados de Amaru en el escenario de pruebas de [Asadujjaman et al., 2018].

tiempo de recuperación de 3.8ms, lo que supone una reducción del tiempo de recuperación de un 15 % aproximadamente por parte de Amaru. Esta diferencia se debe a que la propuesta [Asadujjaman et al., 2018] necesita ir señalizando en cada nodo atravesado el fallo, mientras que Amaru conoce el camino gracias a sus sistema de aprendizaje de etiquetas, lo que hace que los switches solamente necesiten conmutar a la ruta alternativa.

No obstante, también puede darse el caso extremo en el que no se pueda restaurar una ruta porque el camino de backup está afectado por otro fallo. Bajo esta hipotética situación la solución pasa por ejecutar de nuevo Amaru, ya que ninguna de las AMAC aprendidas son válidas, por lo que el tiempo de recuperación se corresponde con el tiempo de convergencia del protocolo. Para la topología sintética de 10 nodos el tiempo de convergencia de Amaru es de 125μs, mientras que en la topología Rogers ese tiempo asciende hasta 1.9ms. En ambos casos el tiempo de convergencia de Amaru se sitúa muy por debajo del umbral máximo de 50ms exigido para recuperar el sistema, por lo que Amaru sigue garantizando, en el peor de los casos, la recuperación del sistema.

Capítulo 5

Protocolos de descubrimiento de caminos múltiples disjuntos

Este capítulo recoge la segunda aportación de la Tesis, centrada en el diseño de protocolos distribuidos que buscan múltiples caminos disjuntos en redes SDN híbridas. Como se comentó en el Capítulo 2, las redes híbridas aúnan los beneficios de las redes SDN (gestión centralizada) y las ventajas de las redes distribuidas (crecimiento escalable e incremento de la fiabilidad de los procesos). El resultado es una red que libera de carga computacional al controlador al distribuir los procesos entre los dispositivos de la capa de infraestructura, incrementando la escalabilidad de la red, ya que un único controlador con los mismos recursos hardware puede gestionar más dispositivos de red al estar sometido a menor carga computacional. Además, este enfoque híbrido incrementa la fiabilidad debido, precisamente, a la distribución del cómputo entre los dispositivos de la capa de infraestructura, garantizando la continuidad del proceso en ejecución aunque algunos de los dispositivos se vean afectados por un fallo.

El mayor problema de la búsqueda de caminos múltiples en redes SDN radica en la forma en la que se efectúa, ejecutando un algoritmo de forma centralizada en el controlador. Estos algoritmos incluyen una operativa matemática compleja que supone una carga computacional apreciable para el controlador, lo que limita la escalabilidad de las propuestas a redes con un tamaño mediano o pequeño. Sin embargo, con Amaru se vio que es posible generar procesos distribuidos de búsqueda de múltiples caminos, rápidos y sencillos, garantizando la escalabilidad en redes de gran tamaño. Por este motivo, se aprovechan las ventajas de las redes SDN híbridas para desarrollar protocolos distribuidos que descubran múltiples caminos y reporten los resultados obtenidos al controlador, reduciendo su carga computacional al delegar el proceso de búsqueda de caminos a la capa de infraestructura. Además, esta aportación añade la disjuntividad entre caminos como valor añadido, reforzando los beneficios que ya de por sí aporta la multiplicidad de caminos, incluyendo dos modalidades de caminos disjuntos (en nodos y en enlaces), característica poco habitual en los trabajos analizados en el Capítulo 2.

Partiendo de esta base, se han desarrollado dos protocolos de descubrimiento de caminos múltiples disjuntos para redes SDN híbridas que son capaces de obtener caminos disjuntos en enlaces y caminos disjuntos en nodos. Ambos protocolos, al igual que Amaru, utilizan técnicas de exploración de red basadas en la familia de protocolos All-Path [Ibáñez & Rojas, 2013, Rojas et al., 2011, Rojas et al., 2015], sin embargo, la aplicación final de los protocolos no está definida como en Amaru, sino que se deja libre (balanceo de carga, tareas de respaldo, seguridad, etc.). El primer protocolo, I-MDP, consta de un proceso secuencial que busca caminos disjuntos entre un par de switches mediante sucesivas exploraciones de la red, mientras que el segundo protocolo, 1S-MDP, optimiza el proceso realizado por su predecesor, siendo capaz de descubrir múltiples caminos disjuntos entre múltiples pares de switches con un único proceso de exploración.

El capítulo comienza mostrando la estructura de los mensajes que utilizan ambos protocolos para descubrir los múltiples caminos disjuntos y, después, describe en detalle cada uno de los protocolos. Tras la descripción de la operativa, se muestra como el controlador recopila y almacena los caminos disjuntos para utilizarlos por los servicios que le demande la red. Por último, se muestra el proceso de implementación de ambos protocolos en el switch BOFUSS y se evalúan ambas propuestas.

5.1 Estructura del paquete

Los switches SDN son dispositivos de red pertenecientes a la capa 2 del modelo Open System Interconnection (OSI) [ITU-T, 1994], por lo que utilizan la trama Ethernet, definida en el estándar IEEE 802.3 [IEEE, 2018], como formato normalizado de paquete. Los protocolos diseñados en este capítulo se adaptan a la trama Ethernet para minimizar el impacto sobre la infraestructura existente, utilizando los campos de las direcciones MAC para identificar los switches entre los que se quieren obtener los caminos disjuntos, y el campo *EtherType* para especificar que el paquete pertenece a la familia de protocolos de descubrimiento de caminos. Además, en el campo de *DATOS* de la trama Ethernet se incluye información de los elementos (switches) que componen el camino disjunto en una nueva estructura de datos, la Estructura Básica de Información por Dispositivo (EBID), cuyo formato está representado en la Figura 5.1. Esta estructura contiene el identificador del switch en la red SDN (*SW ID*), y las interfaces del switch que utiliza el camino disjunto (*INTERFAZ 1* e *INTERFAZ 2*), cuyos tamaños no se eligieron al azar, sino

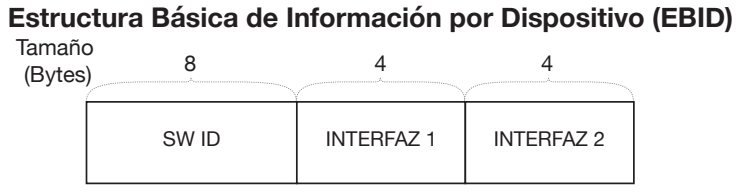


Figura 5.1: Estructura para el descubrimiento de múltiples caminos disjuntos.

que se fijaron respetando el tamaño de las estructuras internas del switch BOFUSS en el que se implementan ambos protocolos. No obstante, el campo de *DATOS* de la trama Ethernet, además de incluir el camino disjunto como una sucesión de EBIDs, también incluye información relativa al protocolo. Una muestra del aspecto que tiene la trama Ethernet dedicada al descubrimiento de caminos disjuntos con I-MDP o 1S-MDP, se puede ver en la Figura 5.2. La trama contiene las direcciones MAC origen y destino de los switches entre los que se están descubriendo los caminos disjuntos, así como el *Ethertype* asignado a los protocolos de descubrimiento de múltiples caminos disjuntos. En el campo *DATOS*, el primer *byte* se reserva para identificar el protocolo (I-MDP o 1S-MDP), el tipo de caminos a descubrir (disjuntos en enlaces o en nodos), y la fase en la que se encuentra el protocolo (descubrimiento topológico o construcción de caminos). El segundo *byte* contiene un número que identifica de forma inequívoca a cada camino disjunto entre un par de switches, y los diez siguientes *bytes* se reservan para operaciones que no están contempladas en el diseño original del protocolo. El resto del campo *DATOS* se dedica exclusivamente a almacenar la lista de EBIDs que describen el camino disjunto. Como se verá más adelante, esta lista se construye de forma distribuida, siendo los switches que componen un camino disjunto, los encargados de introducir cada uno su EBID. No obstante, el campo *DATOS* debe tener una longitud mínima de 46 *bytes* para que funcione la detección de colisiones en el medio físico, como indica el estándar, por lo que si en una trama el campo *DATOS* ocupa menos de 46 *bytes*, la parte final se rellena con ceros hasta alcanzar la longitud mínima. Al igual que el estándar define una longitud mínima de 46 *bytes*, también establece una longitud máxima de 1500 *bytes*, lo que limita el número de EBIDs almacenadas en la trama y, por ende, el número máximo de saltos que puede tener un camino disjunto. Aún así, a pesar de las limitaciones de tamaño, los protocolos de descubrimiento de caminos disjuntos pueden descubrir caminos con una longitud máxima de 93 saltos, como muestra la ecuación 5.1.

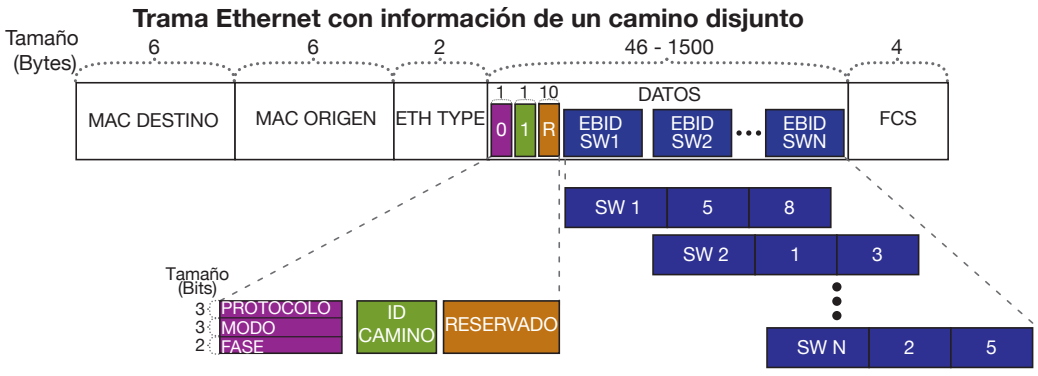


Figura 5.2: Ejemplo de trama Ethernet que contiene información de un camino disjunto.

$$Longitud\ máxima = \frac{Tamaño\ campo\ DATOS - Tamaño\ informacion\ fija}{Tamaño\ EBID} \tag{5.1}$$

$$Longitud\ máxima = \frac{1500\ bytes - 12\ bytes}{16\ bytes} = 93 saltos$$

Estos 93 saltos son más que suficiente para la mayoría de redes SDN, ya que en redes de centros de datos la longitud media de los caminos se sitúa en el rango de los cuatro a los seis saltos [Yao et al., 2014], y en redes que abarcan grandes superficies de terreno, como son las redes WAN, la longitud media se sitúa entre los cinco y los nueve saltos [Soliman et al., 2014]. Por lo tanto, la estructura de trama diseñada permite operar a los protocolos I-MDP y 1S-MDP con la estructura definida por el estándar 802.3 sin introducir cambios sustanciales que afecten a la infraestructura existente, proporcionando además suficiente margen de operación como para obtener caminos de hasta 93 saltos.

5.2 Iterative Multiple Disjoint Paths (I-MDP)

I-MDP es un protocolo que trabaja en redes SDN híbridas de forma distribuida, diseñado para descubrir múltiples caminos disjuntos entre un par de switches mediante un proceso iterativo basado en las técnicas de exploración de red.

En una red SDN estándar, el controlador de la red conoce a todos los dispositivos bajo su domino, ya que previamente ha tenido que establecer la comunicación del plano de control con cada uno de ellos para poder gestionarlos. Sin embargo, no conoce las conexiones entre los dispositivos, a no ser que ejecute un proceso de descubrimiento topológico dedicado para tal fin. Una vez que el controlador conoce la topología, para poder encaminar el tráfico que transcurre por la red, el controlador tiene que ejecutar otro proceso adicional que calcule los caminos disponibles entre los switches origen y destino utilizando la información topológica obtenida, lo que incrementa la carga computacional en el controlador y alarga los tiempos de obtención de las rutas al depender de dos procesos independientes y una computación centralizada.

I-MDP pretende simplificar la búsqueda de caminos utilizando un enfoque SDN híbrido que elimina la etapa de descubrimiento topológico y se centra únicamente en el proceso de búsqueda de rutas entre los switches de la red. Sin embargo, apuesta por realizar este proceso entre los dispositivos de la capa de infraestructura de forma distribuida, prescindiendo del proceso de descubrimiento topológico, y reduciendo la carga computacional en el controlador. Al igual que en las redes SDN, en las redes SDN híbridas el controlador conoce a todos los dispositivos que están bajo su dominio, por lo que él será el encargado de indicar al protocolo I-MDP que busque múltiples caminos disjuntos entre el par de switches origen-destino deseado, en el momento que un servicio de red

lo demande. Cuando termina el proceso de búsqueda, el dispositivo de red designado devuelve los múltiples caminos disjuntos descubiertos al controlador para que éste pueda utilizarlos en los servicios que lo necesiten. El proceso de búsqueda puede hacerse de forma proactiva (se inicia la búsqueda de caminos disjuntos antes de que se produzca la demanda), o reactiva (la búsqueda se inicia cuando un servicio demanda múltiples caminos disjuntos), y será el gestor de la red quien determine si se utiliza un modelo proactivo o reactivo.

I-MDP emplea un proceso de búsqueda estructurado en dos fases que obtiene un camino disjunto entre un par de switches cada vez que se ejecuta. Este proceso se repite tantas veces como posibles caminos disjuntos existan entre el par de switches origen-destino seleccionado, pudiéndose, además, limitar el número de intentos en caso de que se necesiten un número finito de caminos disjuntos. La primera fase realiza un proceso de exploración de la red que genera información topológica relevante para la búsqueda de los caminos disjuntos, mientas que la segunda fase encuentra dichos caminos con la información generada previamente. Para describir con mayor nivel de detalle todo el proceso, se dedica un apartado específico a cada una de las fases, y se adjuntan dos figuras que ilustran el procedimiento de búsqueda en los dos modos de operación definidos: modo caminos disjuntos en enlaces (Figura 5.4) , y modo camino disjuntos en nodos (Figura 5.5).

5.2.1 Primera fase: exploración de la red

La primera fase de I-MDP se inspira en las técnicas de exploración de red de la familia All-Path para examinar la topología y poder descubrir los caminos disjuntos. Este mecanismo de exploración inicia una difusión controlada de mensajes en el switch origen que atraviesa toda la red, hasta alcanzar el switch destino, generando, a su paso, información topológica relacionada con la latencia que presentan los enlaces de la red recorridos. Este proceso de difusión busca el camino más rápido, por lo que los switches intermedios, entre los switches origen y destino, anotan la interfaz por la que reciben el primer mensaje de exploración y lo reenvían a través de todas sus interfaces, menos por la interfaz de entrada, descartando además, todas las copias tardías del mensaje de exploración. El resultado es un árbol de mínima latencia entre el switch origen y el resto de los switches de la red, ya que cada switch anotó la interfaz por la que recibió el primer mensaje de exploración (el más rápido). El estado que guarda cada switch con este proceso es mínimo, ya que solamente tiene que almacenar la interfaz por la que se recibe la primera copia del paquete de exploración recibido, y asociarla con el par de switches origen-destino entre los que se obtienen los caminos disjuntos.

No obstante, la orden que establece el inicio del proceso de búsqueda no reside en el switch origen, sino en el controlador, ya que éste es el único elemento de la red que conoce a todos los dispositivos de la misma y, además, también conoce la demanda global de servicios o recursos de la red como, por ejemplo, la necesidad de obtener múltiples caminos

disjuntos entre dos switches de la red. Por ello, cuando existe una demanda de caminos disjuntos entre dos switches, el controlador le indica al switch origen que inicie el proceso de búsqueda de caminos disjuntos con el switch destino proporcionándole, al switch origen, la dirección del switch destino con el que tiene que encontrar los caminos, y el tipo de caminos a descubrir (disjuntos en enlaces o disjuntos en nodos). El switch origen, tras recibir la orden, genera el paquete de exploración, e incluye en dicho paquete las direcciones MAC origen y destino, el *EtherType* asignado a los protocolos de descubrimiento de caminos disjuntos, y los campos específicos de I-MDP (protocolo, modo, fase e Id del camino). La Figura 5.3 ilustra el inicio de la fase de exploración sobre una topología de ejemplo, donde el controlador, a través del plano de control, le indica al switch *S1* que tiene que buscar caminos disjuntos con el switch *S6*, enviándole la dirección MAC del switch destino, el protocolo a utilizar (I-MDP) y el modo de trabajo (disjunto en enlaces). El switch *S1*, al recibir la información, crea un paquete de exploración con los datos proporcionados por el controlador, añade el identificador único de camino, rellena con ceros el campo *DATOS* del paquete, para alcanzar la longitud mínima fijada por el estándar 802.3, y lo reenvía a través de todas sus interfaces.

A partir de ese momento, el proceso de difusión continúa en los switches intermedios (aquellos que no son ni el switch origen, ni el switch destino), que difunden selectivamente el mensaje de exploración para que llegue al switch destino. La fase de exploración al completo está representada en la mitad izquierda de las Figuras 5.4 y 5.5, en las que cada fila muestra el proceso completo de búsqueda de un camino disjunto. Centrando la atención en la parte izquierda de las figuras, se observa como los switches intermedios solamente aceptan la primera copia recibida del mensaje de exploración (flecha con línea continua), marcan la interfaz por la que entró dicho paquete con un punto de color, y reenvían el paquete por todas las interfaces, excepto por la interfaz de entrada, rechazando además el resto de paquetes (las copias tardías), representados con flechas discontinuas.

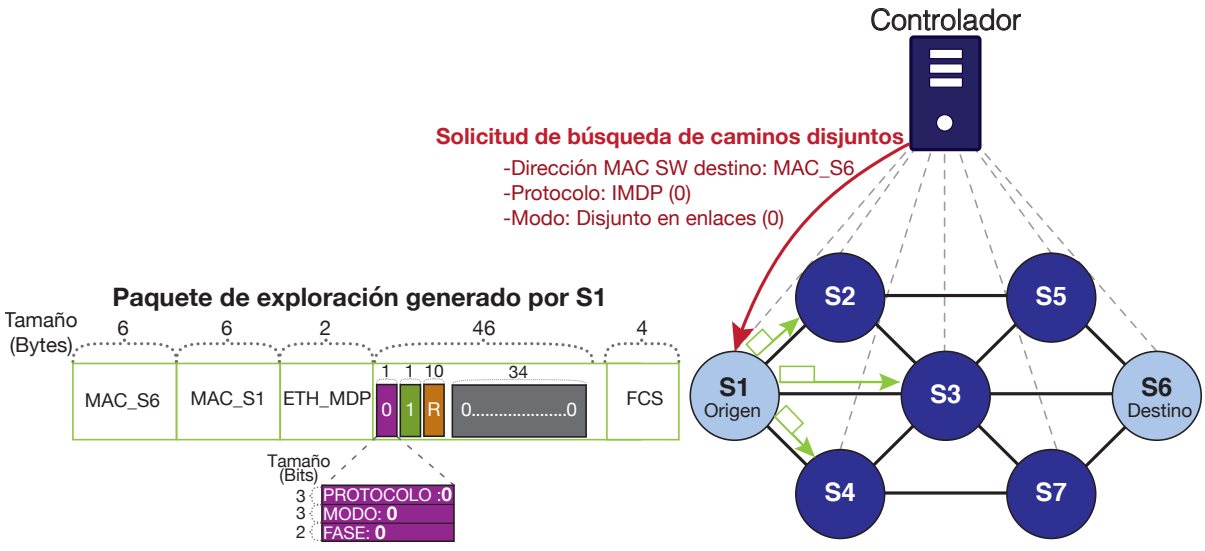
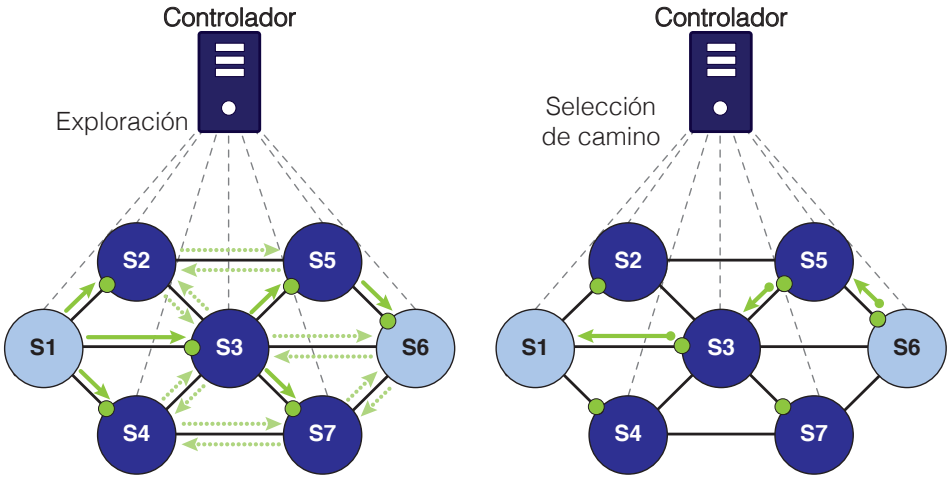


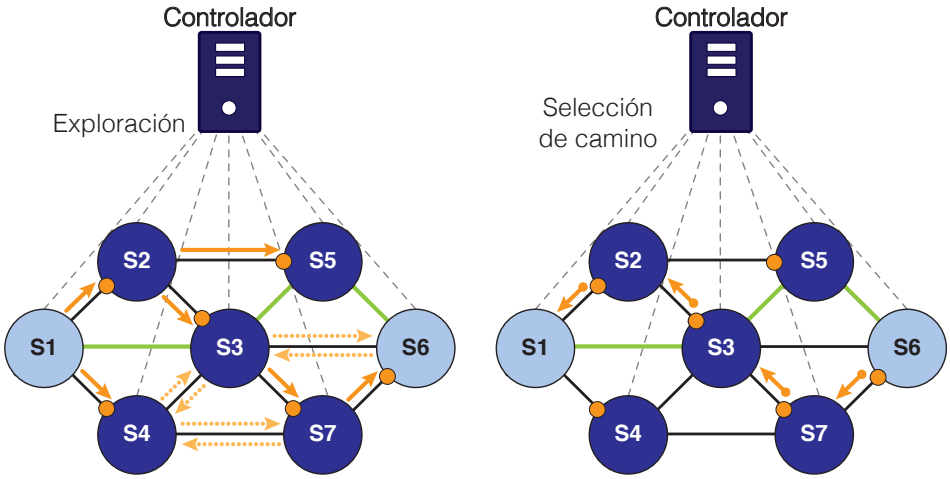
Figura 5.3: Inicio del proceso de exploración en I-MDP.

No obstante, este proceso de exploración presenta sutiles diferencias en función del modo de operación del protocolo (disjunto en enlaces o disjunto en nodos), a partir de la fase de exploración del segundo camino. La parte izquierda de la Figura 5.4b muestra el proceso de exploración para el segundo camino en el modo caminos disjunto en enlaces, apreciándose como el switch *S3* recibe la primera copia del mensaje de exploración, anota la interfaz de entrada y reenvía el mensaje de exploración por todas sus interfaces excepto por la interfaz de entrada del paquete y por las interfaces asociadas al primer camino, garantizando así que los enlaces que pertenecen al primer camino no son explorados, y por lo tanto, no son elegibles para el segundo camino. Si se observa el mismo switch en el proceso de exploración del segundo camino, pero en modo disjunto en nodos (Figura 5.5b), se ve que el *S3* rechaza todos los mensajes de exploración recibidos, debido a que dicho switch se está utilizando en el primer camino disjunto y no puede ser seleccionado para otros caminos. Por lo tanto, el proceso de reenvío de las tramas de exploración es más restrictivo en modo disjunto en nodos, ya que si un nodo es utilizado por un camino previo, éste no reenvía ninguna trama de exploración, mientras que en modo enlace se reenvían las tramas de exploración por todas las interfaces que no tengan un camino previamente descubierto. Por último, las Figuras 5.4c y 5.5c muestran el mismo proceso para el tercer camino disjunto.

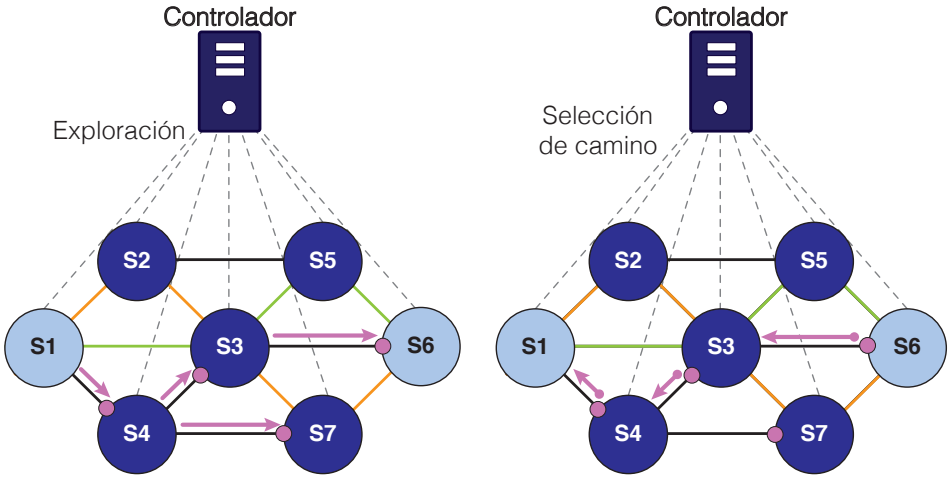
En todas las situaciones, cuando el primer mensaje de exploración alcanza el switch destino, se inicia el proceso de confirmación explicado en el apartado 5.2.2, y cuando éste termina, se inicia una nueva búsqueda de camino ejecutando una nueva exploración de red. Los procesos de exploración posteriores al primer camino se inician de forma periódica, utilizando temporizadores que orquestan de una forma sencilla el borrado de la información obsoleta y el lanzamiento de nuevas exploraciones. Esta técnica garantiza la independencia entre procesos de exploración sin incluir sofisticados procesos síncronos de intercambio de paquetes que aumenten la complejidad del protocolo. Bajo estas premisas se crean dos temporizadores, uno para el borrado de la información, y otro para el lanzamiento de los procesos de exploración. El temporizador de borrado es ligeramente inferior al temporizador de lanzamiento, asegurando que cada proceso de exploración no se encuentra información del proceso de exploración del camino previo. No obstante, este enfoque presenta inconvenientes, ya que el tiempo de convergencia del protocolo depende de la red, por lo que se tienen que fijar los temporizadores en función del RTT de la red. Esta medida estima el tiempo de ida y vuelta de un paquete entre dos puntos de la red, y depende, entre otros parámetros, del tamaño de la red, ya que no es lo mismo una red WAN desplegada a lo largo de cientos o miles de kilómetros, que una red de centro de datos, cuyo despliegue está concentrado en un edificio. Además, el valor de estos temporizadores se debe incluir en cada uno de los switches, trabajo que tradicionalmente realizaba el administrador de la red manualmente, sin embargo, al trabajar con redes SDN este trabajo se puede hacer de forma centralizada desde el controlador, bien mediante tecnologías como P4, que permiten programar desde el controlador el comportamiento de



a) Procesos de exploración y de selección del primer camino.



b) Procesos de exploración y de selección del segundo camino.



c) Procesos de exploración y de selección del tercer camino.

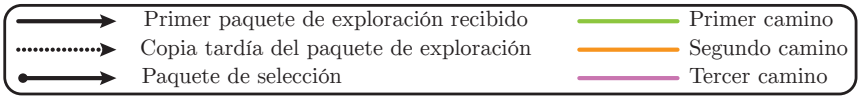
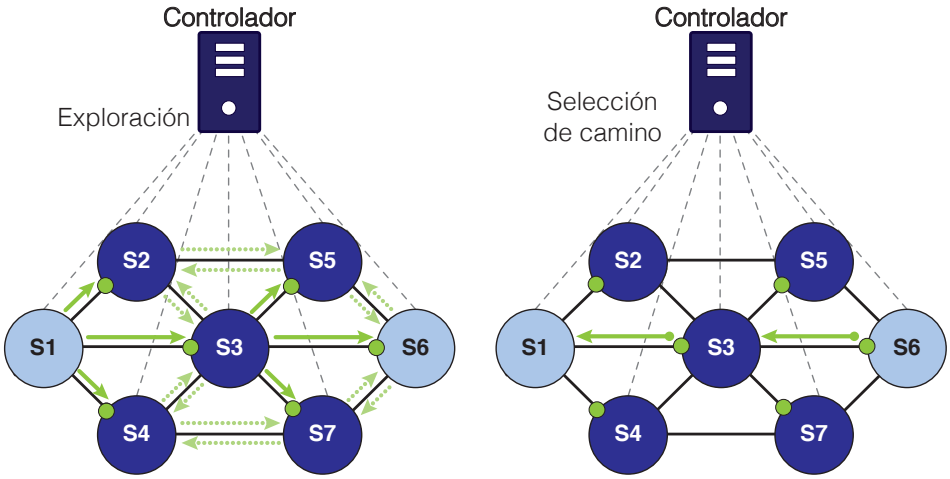
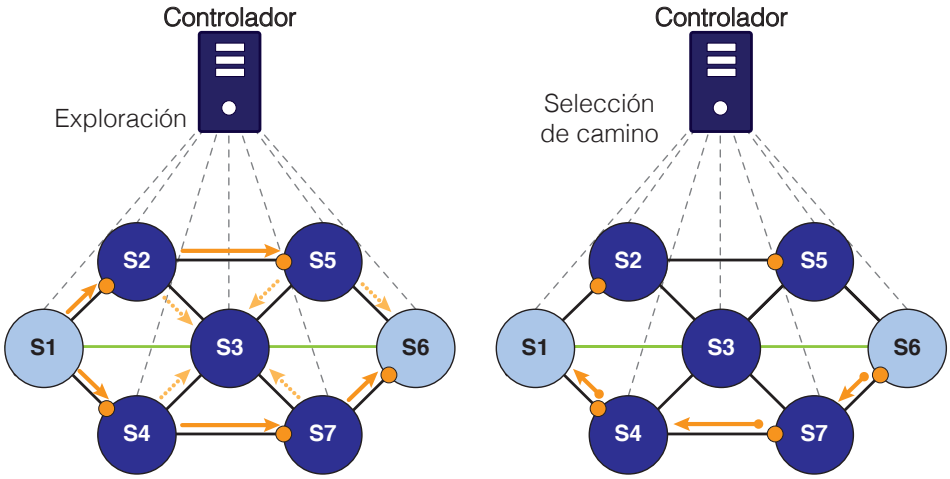


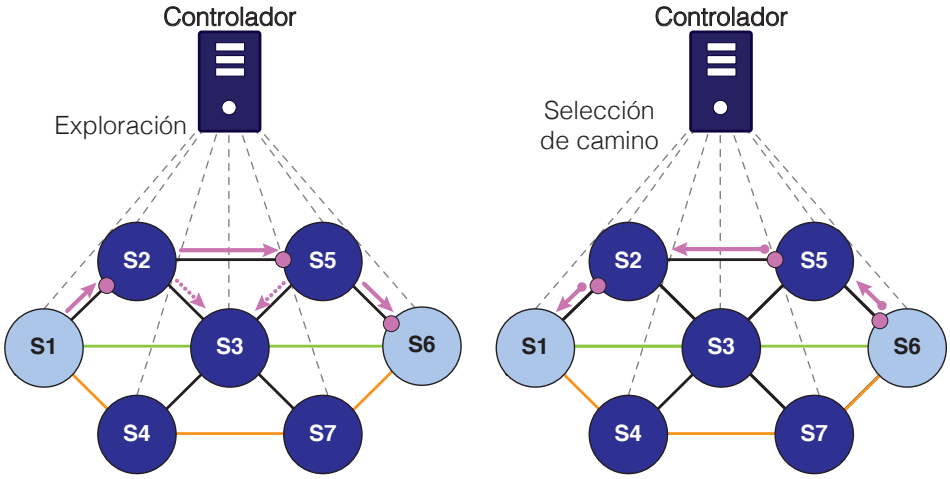
Figura 5.4: Proceso de búsqueda de caminos de I-MDP en modo enlace.



a) Procesos de exploración y de selección del primer camino.



b) Procesos de exploración y de selección del segundo camino.



c) Procesos de exploración y de selección del tercer camino.

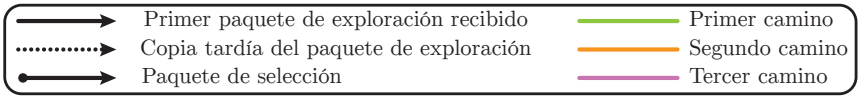
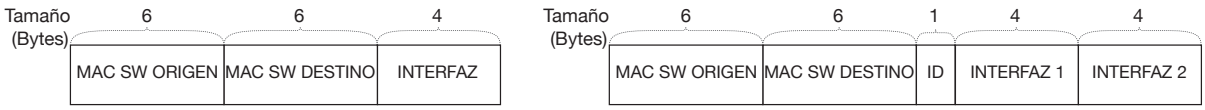


Figura 5.5: Proceso de búsqueda de caminos de I-MDP en modo nodo.

los switches, bien programando estos eventos periódicos en el controlador para que sea éste quien envíe las acciones de borrado y de lanzamiento a los switches.

Por último, hay que mencionar que la búsqueda de caminos disjuntos entre un par de switches termina cuando el switch origen ha ejecutado tantos procesos de exploración como interfaces tiene. De esta forma, se garantiza que se buscan todos los posibles caminos disjuntos entre dicho par de switches.

En cuanto a las estructuras de datos que manejan los switches, éstas se reducen a dos tablas, una primera tabla que almacena la información generada en la fase de exploración, representada en la Figura 5.6a, y una segunda tabla que almacena información relativa a los caminos descubiertos, mostrada en la Figura 5.6b. Cada uno de los switches de la red contiene ambas estructuras de datos y almacena en ellas información de forma local, recopilando únicamente los datos que le conciernen a él. La tabla asociada al proceso de exploración almacena la dirección MAC de los switches origen y destino entre los que se están descubriendo los caminos disjuntos, junto con la interfaz por la que entró el primer paquete de exploración. Con esta información, cada switch es capaz de discriminar copias tardías del paquete de exploración (puesto que comparten las mismas direcciones MAC origen y destino), y hacer los reenvíos selectivos que excluyen a la interfaz por la que entró el primer paquete de exploración. La información almacenada en esta tabla se borra antes de que se ejecute el siguiente proceso de exploración, al saltar el temporizador de borrado explicado anteriormente, puesto que la información generada en cada proceso de exploración solo es válida en dicho proceso. Por ejemplo, en las Figuras 5.4 y 5.5 antes de que se ejecute un nuevo proceso de exploración, la información generada por el proceso de exploración previo se ha borrado de toda la red (círculos de color). En cuanto a la tabla de caminos confirmados, ésta almacena la información que identifica inequívocamente a cada uno de los caminos disjuntos que pasan por el switch, y se rellena en la fase de confirmación de caminos, como se verá en la Sección 5.2.2. Entre los datos guardados están las direcciones MAC de los switches origen y destino, el identificador único de camino para ese par de switches, y las interfaces que utiliza el camino en el switch. Sin embargo, a pesar de que esta tabla se rellena en la fase de confirmación de caminos, su contenido se consulta en la fase de exploración para identificar interfaces utilizadas por caminos previos, y evitar el reenvío de los mensajes de exploración a través de dichas interfaces cuando los caminos son disjuntos en enlaces. Además, también es útil para rechazar cualquier mensaje de exploración en el modo caminos disjuntos en nodos, ya que a través de esta tabla se sabe si el switch contiene un camino disjunto previo entre el



a) Tabla asociada al proceso de exploración. b) Tabla de caminos confirmados.

Figura 5.6: Estructuras de datos utilizadas en I-MDP.

mismo par de switches origen-destino.

5.2.2 Segunda fase: selección de caminos

La segunda fase de I-MDP selecciona el camino de menor latencia disponible entre los switches origen y destino en utilizando la información generada por la fase de exploración. La disjuntividad entre caminos se consigue gracias a la fase de exploración, que excluye de ser nuevamente explorados a los enlaces o switches que son utilizados por otros caminos disjuntos entre el mismo par de switches origen-destino. De esta forma, cada proceso de exploración proporciona el camino más rápido entre los switches origen-destino, al mismo tiempo que garantiza la disjuntividad entre caminos. Por lo tanto, para construir el camino, la fase de selección de caminos solamente tiene que elegir, en cada switch, la interfaz aprendida en la fase de exploración. El inicio de la fase de selección de camino comienza en el switch destino, al recibir el primer paquete de exploración, modificándolo y reenviándolo de vuelta hacia el switch origen. El proceso continúa hasta que el paquete alcanza el switch origen, siendo los switches intermedios los encargados de incluir en el paquete la información sobre el camino en construcción y de reenviar el paquete hacia el switch origen. La mitad derecha de las Figuras 5.4 y 5.5 muestra, a grandes rasgos, el proceso de selección de caminos en los modos caminos disjuntos en enlaces y disjuntos en nodos, respectivamente. Sin embargo, para un análisis más detallado se utiliza la Figura 5.7, que ejemplifica, minuciosamente, el proceso de selección de caminos para el primer camino disjunto entre los switches *S1* y *S6* en el modo caminos disjuntos en enlaces. En dicha figura, el switch destino *S6*, cuando recibe el primer paquete de exploración enviado desde el switch origen *S1* (paquete sin color de fondo), lo modifica para transformarlo en un paquete de selección de camino (paquete con fondo verde). La modificación consiste en cambiar de orden las direcciones MAC origen y destino del paquete (puesto que el nuevo paquete se va a enviar en sentido inverso), actualizar el cambio de fase en el campo del paquete dedicado a tal fin, e incluir la información de la EBID. La EBID del switch destino incluye el identificador del switch y la interfaz de entrada por la que se recibió el paquete de exploración, mientras el campo *INTERFAZ 2* se deja vacío al tratarse de uno de los switches extremos del camino. Cuando ha rellenado todos los campos, el switch *S6* reenvía hacia el switch origen el nuevo paquete a través de la interfaz por la que recibió la primera copia del paquete de exploración. Además, el switch *S6* incluye en la tabla de caminos confirmados las direcciones MAC de los switches origen y destino, el identificador del camino y las interfaces del switch utilizadas por el camino disjunto. Los switches intermedios, al recibir el paquete de confirmación, introducen una nueva entrada en la tabla de caminos confirmados con los datos contenidos en el paquete (las direcciones MAC y el identificador del camino), y las interfaces utilizadas por el camino disjunto en construcción (los círculos marcados en color verde). Cada switch, además, incluye su EBID en el paquete y lo reenvía hacia el switch origen por la interfaz

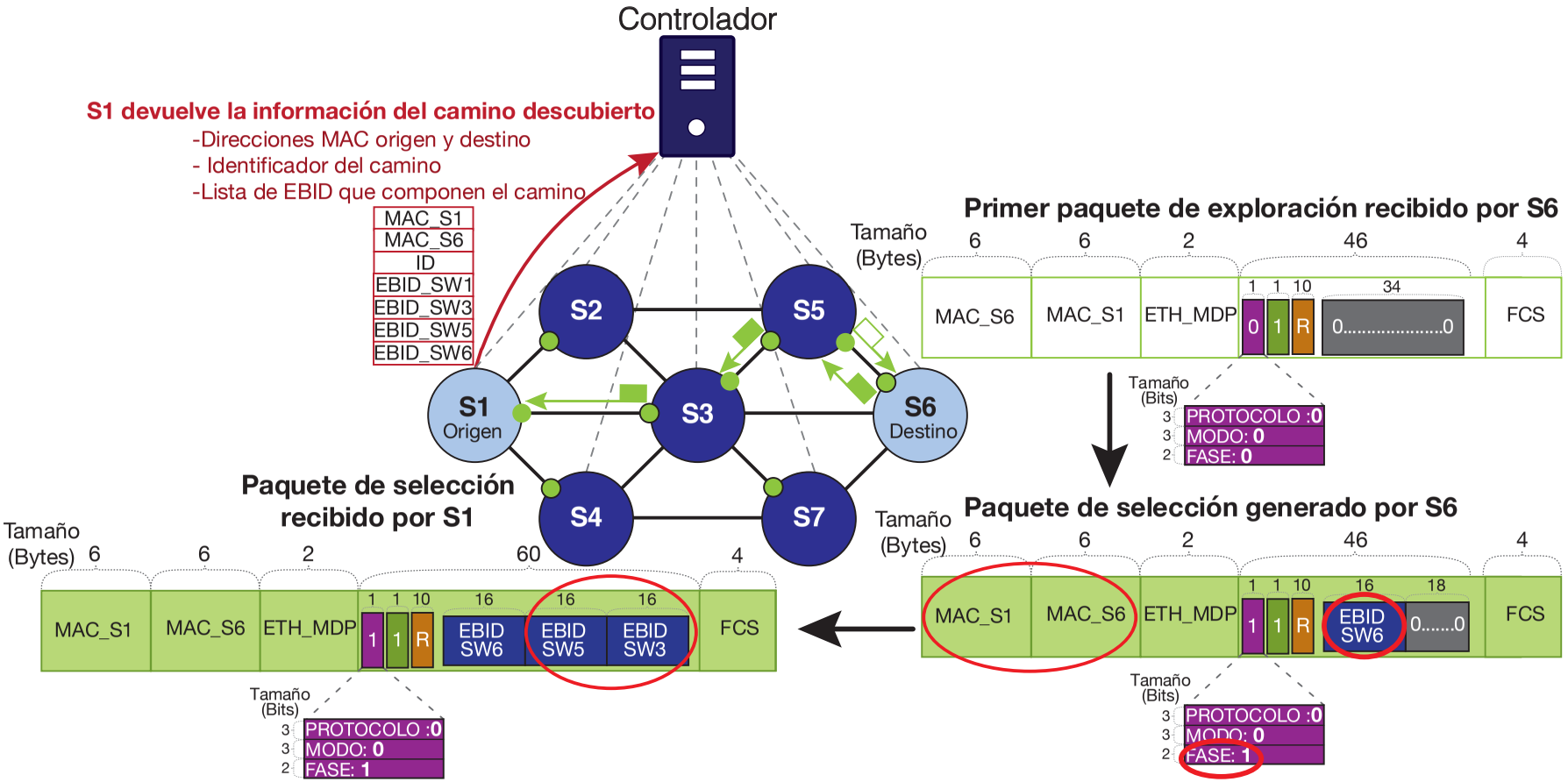


Figura 5.7: Proceso de selección de camino de I-MDP.

aprendida en la fase de exploración (marcada en color verde). Al final, el switch *S1* recibe el paquete de selección de camino, calcula su EBID, y extrae del paquete las EBIDs que describen el camino seleccionado para enviárselo al controlador. De esta forma, el controlador conoce los caminos disjuntos entre los switches *S1* y *S6* sin participar de forma activa en el proceso de búsqueda. Una vez que el controlador ha recibido el camino disjunto, lo almacena en su base de datos para utilizarlo cuando alguno de los servicios de red lo requiera.

El proceso de selección de caminos para el resto de caminos es igual. El switch destino inicia el proceso de selección transformando y reenviando, hacia el switch origen, el paquete de exploración recibido. Los switches intermedios actualizan su tabla de caminos confirmados, incluyen su EBID en el paquete, y lo reenvían hasta el switch origen, que finalmente recopila la información del camino descubierto para enviársela al controlador.

5.2.3 Características adicionales

La técnica de exploración de red utilizada por I-MDP descubre, en cada ejecución, el camino más rápido (de menor latencia) que va desde el switch origen al switch destino, puesto que cada uno de los switches intermedios anota solamente la interfaz que recibe el primer paquete de exploración. Por lo tanto, como cada camino disjunto necesita un proceso de exploración, cada uno de los caminos disjuntos se corresponde con el camino de menor latencia en el momento de la búsqueda. Esta característica es importante, ya que la búsqueda de caminos disjuntos tiene en cuenta la carga de la red en el momento en que se inicia al proceso de descubrimiento de caminos, devolviendo siempre el conjunto de caminos disjuntos con menor latencia en ese momento. Es interesante remarcar que el conjunto de caminos disjuntos devuelto se corresponde con una foto fija de la red en un instante de tiempo determinado, por lo que si se utilizan esos caminos disjuntos pasado un periodo de tiempo prolongado, las condiciones de la red pueden haber cambiado y no corresponderse con los caminos disjuntos de menor latencia. Del mismo modo, esta característica tan peculiar hace que dos ejecuciones del protocolo entre el mismo par de switches origen-destino no obtenga los mismos resultados debido, precisamente, a las condiciones cambiantes de la red.

El diseño del protocolo posibilita el uso de una métrica alternativa a la latencia de la red. Por ejemplo, se puede utilizar el campo reservado del paquete para incluir información sobre el número de saltos que ha atravesado el paquete de exploración, y realizar el aprendizaje con el paquete que contenga el menor número de saltos, en lugar de utilizar como criterio el primer paquete recibido.

Además, I-MDP permite realizar procesos de búsqueda independientes entre distintos

pares de switches origen-destino, por lo que se pueden ejecutar varios procesos de búsqueda de caminos disjuntos entre distintos pares de switches en paralelo. Esta independencia entre procesos se consigue gracias al sistema de identificación único que incluye las direcciones MAC de los switches origen y destino, y el identificador del camino. Con esta información, cada switch puede reconocer el proceso de búsqueda al que pertenece cada paquete y actuar en concordancia. Asimismo, la ejecución de procesos de búsqueda en paralelo es una característica importante que reduce el tiempo de búsqueda de los caminos disjuntos.

5.3 One Shot Multiple Disjoint Paths (1S-MDP)

1S-MDP surge como un protocolo que refuerza el enfoque de las redes SDN híbridas y que mejora debilidades presentes en I-MDP. Este último protocolo es capaz de ejecutar varios procesos de descubrimiento de caminos disjuntos, en paralelo, entre distintos pares de switches. Sin embargo, I-MDP utiliza temporizadores (tiempos de espera) para ejecutar los procesos secuenciales de búsqueda entre caminos del mismo par de switches origen-destino. Estos tiempos de espera ralentizan el proceso de búsqueda, ya que los temporizadores deben estar sobredimensionados para garantizar que no exista solapamiento entre dos procesos de búsqueda, por lo que el descubrimiento de múltiples caminos disjuntos entre el mismo par de switches origen-destino requiere más tiempo del que realmente supone. 1S-MDP va más allá, ya que, en primer lugar, prescinde del uso de temporizadores entre procesos, y elimina la secuencialidad entre procesos de búsqueda, lo que reduce drásticamente el tiempo invertido en la búsqueda de caminos, o tiempo de convergencia. En segundo lugar, 1S-MDP incrementa el ratio de caminos disjuntos descubiertos por cada proceso de exploración, al realizar una búsqueda simultánea de múltiples caminos disjuntos entre múltiples pares de switches. Por todo ello, 1S-MDP es capaz de obtener al mismo tiempo múltiples caminos disjuntos entre múltiples pares de switches, ejecutando solamente un único proceso de búsqueda.

Al igual que I-MDP, 1S-MDP plantea un modelo estructurado en dos fases: la primera fase realiza un proceso de exploración de la red desde un switch origen, y la segunda fase selecciona los múltiples caminos disjuntos. La diferencia radica en que la fase de exploración genera más información topológica en cada uno de los switches que, posteriormente, es aprovechada por la segunda fase para confirmar, de forma simultánea, múltiples caminos disjuntos desde múltiples switches destino. De este modo, 1S-MDP es capaz de obtener múltiples caminos disjuntos entre un switch origen y múltiples switches destino de manera concurrente, exprimiendo la información recopilada en cada proceso de exploración, y eliminando el uso de temporizadores que alargaban los tiempos de espera entre procesos. Los switches que forman parte de los extremos de cada camino disjunto se denominan switches frontera, ya que representan los puntos entre los que se construyen los caminos disjuntos, siendo el administrador de la red quien decide que switches son

frontera y cuales no. Además, 1S-MDP permite limitar la búsqueda a un número máximo de caminos entre cada par de switches origen-destino, en el caso de que la aplicación que demanda el servicio solo necesite un número limitado de caminos disjuntos, y también es capaz de obtener caminos disjuntos en enlaces y caminos disjuntos en nodos. Sin embargo, al contrario que su predecesor, la diferencia entre ambos modos de operación no se encuentra en la fase de exploración, si no en la de confirmación, como se verá más adelante.

Para describir con mayor nivel de profundidad cada una de las fases se va a dedicar un apartado específico a cada una de ellas, al igual que se hizo en la Sección 5.2 con I-MDP.

5.3.1 Primera fase: exploración de la red

Al igual que I-MDP, 1S-MDP se inspira en la técnica de exploración de red utilizada por la familia de protocolos All-Path para generar la información topológica que, posteriormente, se utilizará para seleccionar los caminos disjuntos. El nuevo proceso de exploración difiere ligeramente de su predecesor, ya que ha necesitado pequeñas modificaciones que, sin cambiar su esencia, incrementan la información topológica recopilada para la construcción de caminos disjuntos. Estas modificaciones consisten, por un lado, en reenviar el mensaje de exploración a través de todas las interfaces del switch, incluyendo también la interfaz del switch por la que se recibe el primer paquete de exploración, lo que permite explorar todos los enlaces de la red en ambas direcciones; y, por otro lado, en extraer información temporal de las copias tardías recibidas. Ahora, los switches, antes de descartar las copias tardías, anotan el instante temporal en el que reciben dichas copias, lo asocian con la interfaz que la recibe, y ordenan las interfaces del switch en una lista de menor a mayor latencia. Este ordenamiento temporal será clave en la fase de selección de caminos, dónde cada uno de los switches elegirá la interfaz libre más prioritaria (menor coste temporal).

Una vez vistas, a grandes rasgos, las principales diferencias entre la fase de exploración de I-MDP y 1S-MDP, la Figura 5.8 ayuda a explicar, con mayor nivel de detalle, el proceso que sigue la fase de exploración de 1S-MDP. Como en I-MDP, el controlador de la red indica al switch origen que inicie el proceso de búsqueda, enviándole la dirección MAC del switch o switches destino, el protocolo a utilizar, y el modo de operación. Como 1S-MDP permite una búsqueda de caminos disjuntos entre múltiples pares de switches frontera, el campo de la dirección MAC destino puede variar. Bien puede estar compuesto por la dirección MAC de uno de los switches de la red, si solo se quieren descubrir caminos disjuntos entre el switch origen y un switch destino, bien por una dirección multicast a la que pertenecen un subconjunto del grupo de switches frontera, descubriendo caminos disjuntos entre el switch origen y los switches pertenecientes al grupo multicast, bien por la dirección de broadcast, descubriendo caminos disjuntos entre el switch origen y el resto de switches frontera. Esta es una de las primeras diferencias con respecto a I-MDP ya que, es posible descubrir caminos disjuntos entre múltiples pares de switches

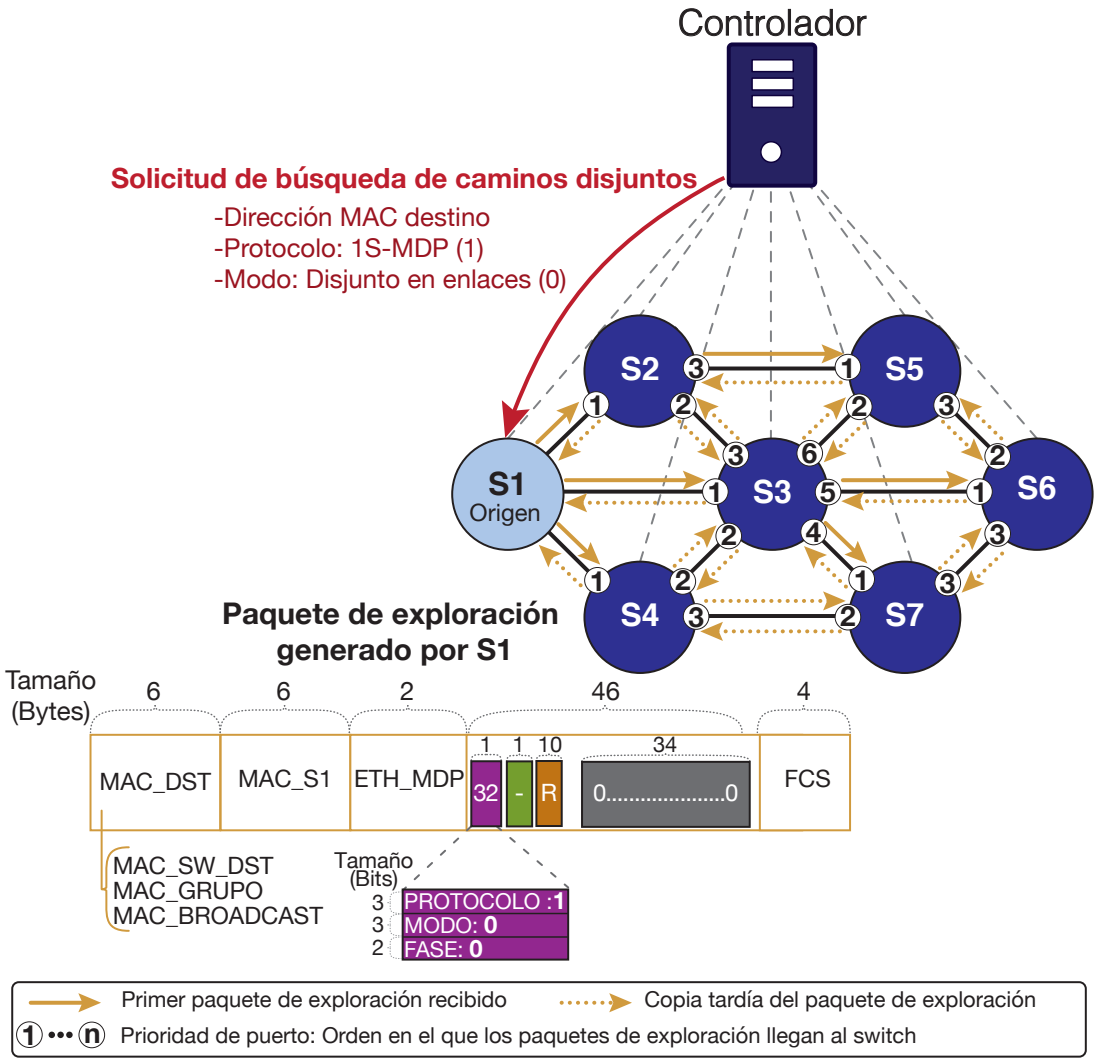


Figura 5.8: Proceso de exploración realizado por 1S-MDP.

(un único origen y múltiples destinos) con un solo proceso de exploración. Una vez que el switch origen (*S1* en la Figura 5.8), recibe la orden del controlador, inicia el proceso de exploración, generando un paquete de exploración y enviándolo a través de todas sus interfaces. En el caso particular de la Figura 5.8, el switch *S1* introduce en el paquete las direcciones MAC del switch *S1* y la dirección de broadcast, el *EtherType* asociado a los protocolos de descubrimiento de caminos disjuntos, y los datos relativos al protocolo 1S-MDP (identificador del protocolo, modo de operación y fase). El resto de switches difunden, a través de todas sus interfaces, la primera copia del paquete de exploración recibida, rechazando el resto de copias tardías para evitar tormentas de mensajes. Sin embargo, ahora los switches anotan el instante de tiempo en el que reciben cada uno de los mensajes de exploración, y ordenan sus interfaces asignándoles un nivel de prioridad en función del instante de llegada de dichos paquetes (mayor prioridad a las interfaces que primero reciben los paquetes). Por ejemplo, el switch *S3* recibe la primera copia a través de la interfaz que le conecta con *S1*, por lo que a dicha interfaz le asigna la prioridad máxima (1), y después reenvía el paquete por todas sus interfaces. Más tarde, el switch *S3*

recibe copias tardías del paquete de exploración que son rechazadas, pero le proporcionan los niveles de prioridad (la segunda copia llega a través de *S4*, la tercera a través de *S2*, etc.). Este mecanismo proporciona información de caminos alternativos al camino de menor latencia desde el switch origen, lo que será de gran utilidad en la fase de selección de caminos. Además, el switch origen rechaza todas las copias del paquete de exploración, puesto que el objetivo de esta fase de exploración es descubrir caminos que parten de un switch origen y llegan al resto de switches.

La estructura de datos utilizada, por la fase de exploración, en cada uno de los switches, es similar a la empleada en el protocolo I-MDP, solo que ahora, por cada proceso de exploración, cada uno de los switches almacena tantas entradas como interfaces tiene. La Figura 5.9 muestra la estructura de la tabla, donde las direcciones MAC origen y destino identifican el proceso de exploración, y el campo interfaz contiene las interfaces del switch ordenadas en función del instante de llegada de los paquetes, estableciendo los niveles de prioridad (la primera interfaz almacenada es la interfaz por la que se recibe el primer paquete de exploración, la segunda entrada se corresponderá con la interfaz que recibe el segundo paquete de exploración, etc.). Este formato de tabla, además, permite simultanear varios procesos de exploración originados desde diferentes switches origen, ya que cada proceso se identifica de forma inequívoca al almacenar la dirección MAC del switch origen y la dirección MAC destino.

Los beneficios que aporta esta nueva técnica de exploración son claros, ya que con un único proceso de difusión se consiguen caracterizar todas las interfaces de cada uno de los switches, asignándoles un nivel de prioridad en función de la latencia que presenta la red desde el switch origen hasta cada interfaz. Esta información permite construir múltiples caminos disjuntos desde múltiples switches atendiendo a un coste de latencias, como se verá en la siguiente sección.

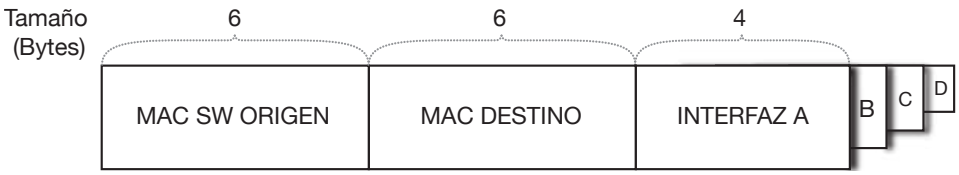
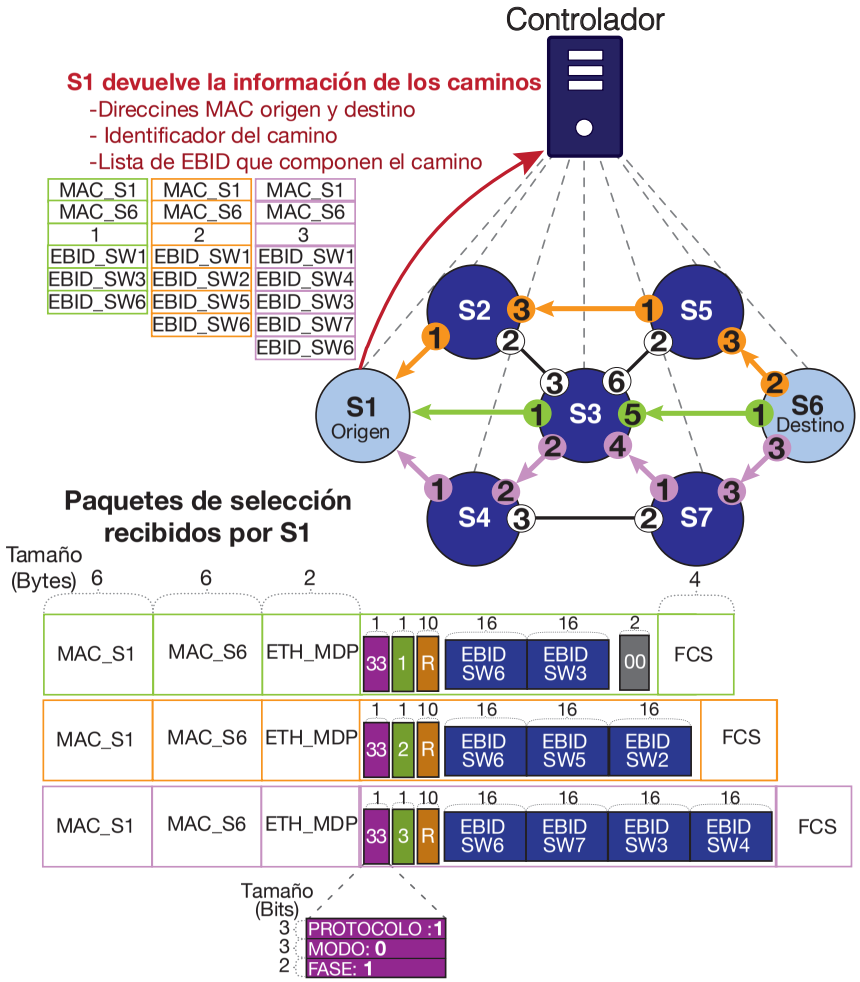


Figura 5.9: Estructura de datos para el proceso de exploración con 1S-MDP.

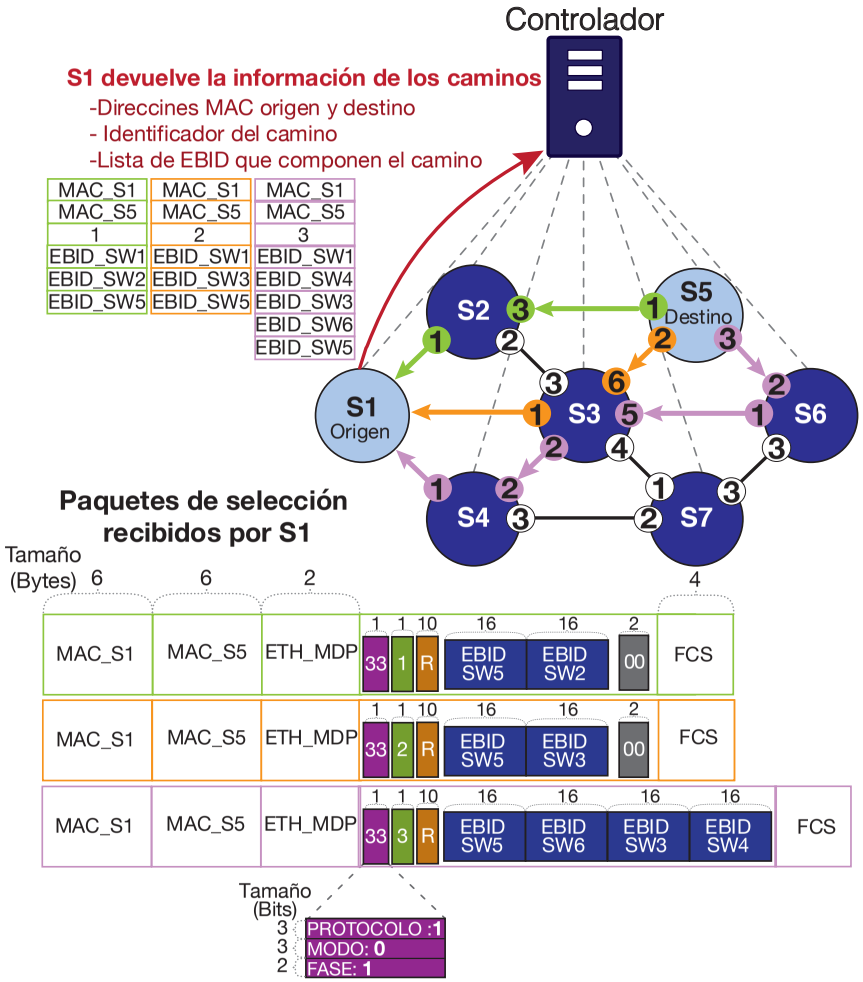
5.3.2 Segunda fase: selección de caminos

En esta segunda fase, 1S-MDP aprovecha la información generada por la fase de exploración para seleccionar los caminos disjuntos, al igual que ocurría en I-MDP. Sin embargo, debido a la información adicional que recoge en la fase de exploración, este protocolo puede seleccionar de manera concurrente múltiples caminos disjuntos entre múltiples pares de switches, eliminando la secuencialidad entre procesos que presentaba I-MDP.

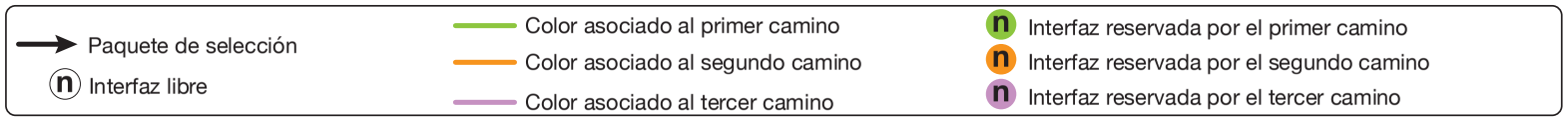
1S-MDP puede realizar la búsqueda de caminos disjuntos entre un switch origen y uno o varios switches destino en función de la dirección MAC incluida en el mensaje de exploración. El escenario elegido para escenificar la fase de selección de camino parte del ejemplo mostrado en la fase de exploración, y cuenta con un conjunto de 3 switches frontera, *S1*, *S5* y *S6*, de los cuales *S1* representa al switch origen, y los otros dos a los switches destino. Este escenario queda ilustrado en la Figura 5.10 para representar el modo disjunto en enlaces, y en la Figura 5.11 para mostrar el modo disjunto en nodos. Ambas figuras se componen, a su vez, de dos subfiguras que describen gráficamente el proceso de selección de caminos desde los switches *S6* y *S5*. Además, las figuras también muestran el aspecto que tiene el paquete de selección de caminos cuando lo recibe el switch origen. A pesar de que el proceso de selección de caminos de los switches *S5* y *S6* está representado en figuras separadas, estos procesos ocurren de forma simultánea en el tiempo, al igual que ocurre con los procesos de selección de caminos disjuntos del mismo par de switches origen-destino, lo que reduce drásticamente el tiempo que tarda el protocolo en obtener el conjunto de caminos disjuntos. El inicio del proceso de selección de caminos en los switches frontera destino es similar al de I-MDP, solo que ahora, los switches frontera destino, en lugar de iniciar el proceso de selección de caminos solamente con el primer mensaje de exploración recibido, lo inician al recibir cualquier mensaje de exploración, independientemente de su orden de llegada. Para ello, los switches frontera comprueban que la dirección MAC del paquete de exploración coincide, bien con su con su dirección MAC, bien con la dirección multicast asociada a su grupo, bien con la dirección de broadcast y, en caso afirmativo, generan el mensaje de selección de camino y lo envían hacia el switch origen a través de la interfaz por la que recibieron el paquete de exploración. En el ejemplo de la Figura 5.10a, cuando el switch *S6* recibe el primer paquete de exploración a través de la interfaz marcada con un 1 (color verde), procede a generar el paquete de selección de camino, transformando el paquete de exploración recibido. Para ello incluye en el campo del paquete dirección MAC origen su dirección MAC, y en el campo dirección MAC destino la dirección MAC del switch *S1*, añade el identificador del camino, actualiza el campo fase del protocolo, e incluye su EBID. El identificador de camino se genera atendiendo al orden de generación de caminos entre el par de switches origen-destino (al primer camino generado se le asigna un 1, al segundo un 2, etc.). Sin embargo, se puede elegir cualquier otro sistema de numeración siempre que se



a) Selección de caminos entre S1 y S6.

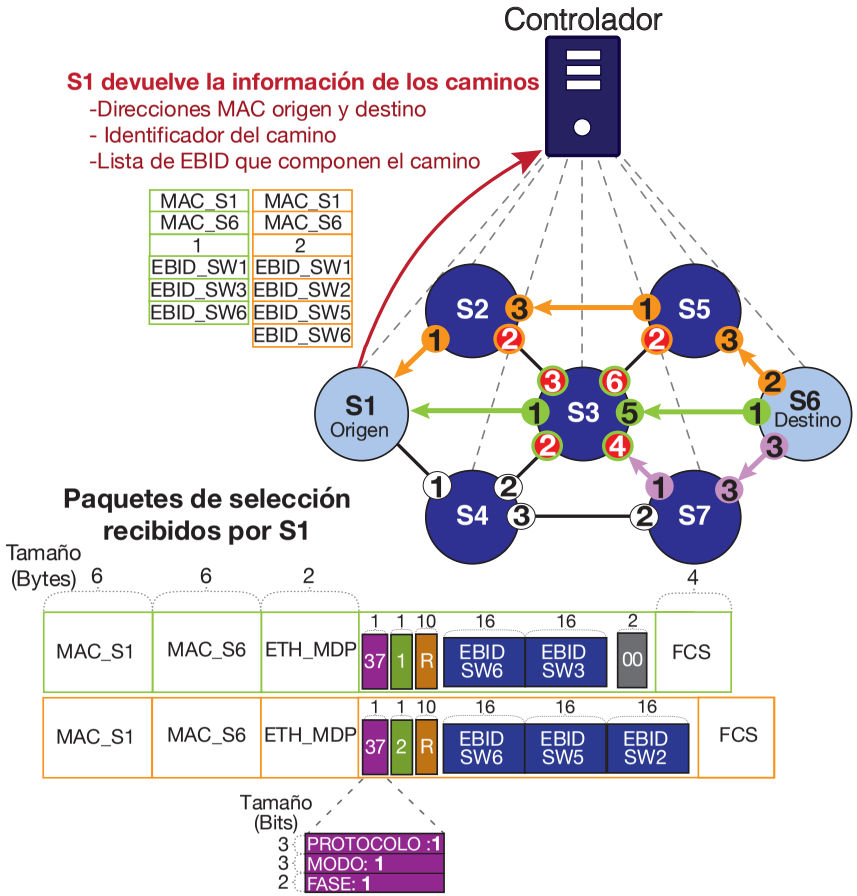


b) Selección de caminos entre S1 y S5.

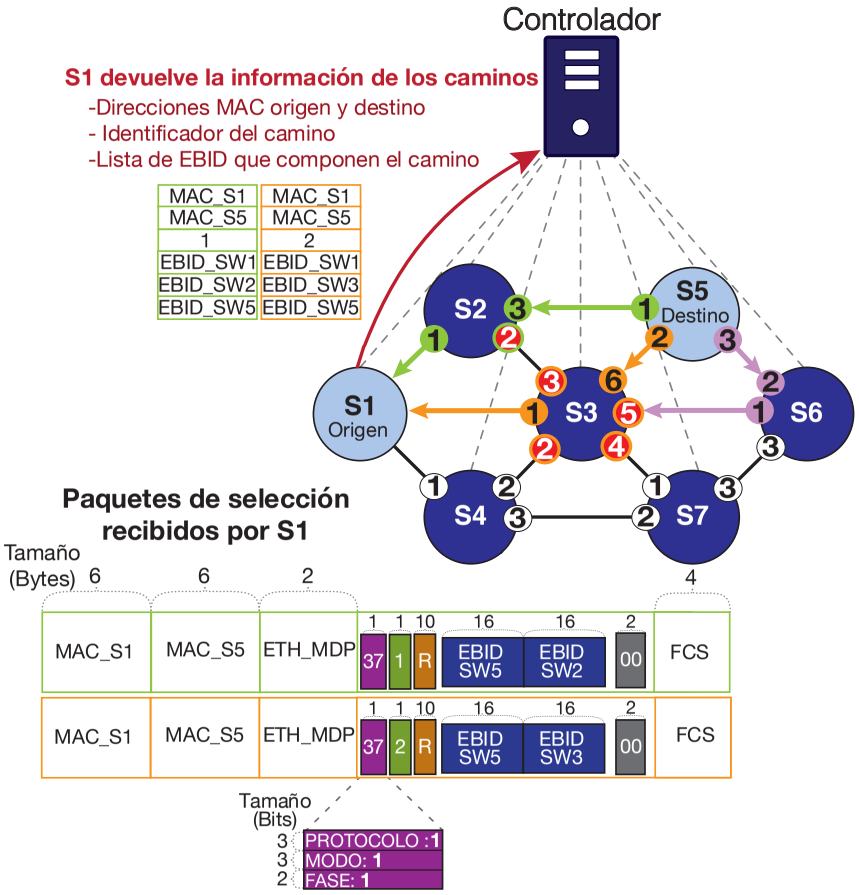


c) Leyenda.

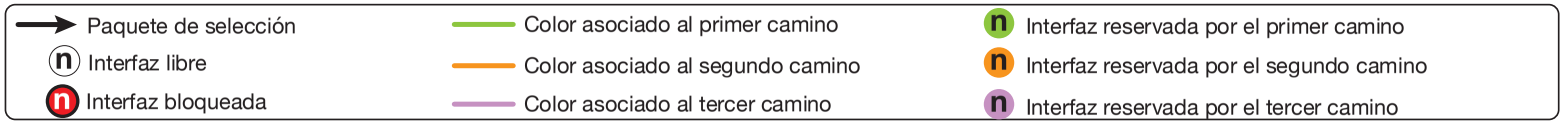
Figura 5.10: Proceso de selección de caminos disjuntos en enlaces.



a) Selección de caminos entre S1 y S6.



b) Selección de caminos entre S1 y S5.



c) Leyenda.

Figura 5.11: Proceso de selección de caminos disjuntos en nodos.

garantice que cada proceso de selección de caminos entre un par de switches es único. Una vez generado el paquete, el switch *S6* lo envía hacia el switch origen a través la interfaz por la que recibió el paquete de exploración, repitiéndose el mismo proceso cuando *S6* recibe la segunda y la tercera copia del mensaje de exploración. Los switches intermedios, cuando reciben un mensaje de selección de camino, buscan en la tabla de exploración la interfaz de mayor prioridad libre (interfaz no utilizada por otros caminos disjuntos entre el mismo par de switches origen-destino), introducen su EBID en el paquete, y reenvían dicho paquete por la interfaz (libre) seleccionada. Este método de búsqueda en la tabla de exploración es el que garantiza la disjuntividad de caminos, al no elegir interfaces o switches que están ocupados por otros caminos. En las figuras de ejemplo las interfaces utilizadas por un camino disjunto están marcadas en el color del camino para indicar que no están disponibles para otros caminos disjuntos. Por ejemplo, en la Figura 5.10a, en el proceso de selección del tercer camino, cuando el switch *S3* recibe el mensaje de selección de camino, éste elige la interfaz marcada con un 2, puesto que es la interfaz con mayor prioridad que tiene disponible en ese momento, ya que la interfaz marcada con un 1 fue elegida por otro camino disjunto. Al final, el switch origen recibe los paquetes de selección de camino que incluyen las EBIDs de los switches que componen el camino disjunto, desempaqueta dicha información y la envía al controlador.

El proceso de selección de caminos iniciado desde el switch *S5* (Figura 5.10b) es exactamente igual que el proceso descrito para el switch *S6*. *S5* genera un mensaje de selección de camino por cada paquete de exploración recibido y lo envía hacia el switch origen atravesando switches intermedios. Los switches intermedios eligen las interfaces libres con mayor prioridad y añaden su EBID al paquete. Finalmente, los mensajes llegan al switch origen, quien envía la composición del camino disjunto al controlador.

La Figura 5.11 muestra el proceso de selección de caminos para el modo disjunto en nodos, siguiendo la misma estructura que la Figura 5.10, exponiendo el proceso de selección de caminos del switch *S6* en la mitad izquierda de la figura, y en la mitad derecha, el proceso de selección de caminos del switch *S5*. Sin embargo, se puede observar como el modo disjunto en nodos solamente consigue dos caminos disjuntos por cada switch destino, frente a los 3 que consigue el modo disjunto en enlaces debido, precisamente, a las restricciones impuestas por el modo disjunto en nodos. Si bien el comportamiento en los switches frontera destino es igual que para el modo disjunto en enlaces (por cada paquete de exploración recibido se genera un paquete de selección de camino), no ocurre lo mismo con el resto de switches. Ahora, los switches que no pertenecen a los extremos del camino solo pueden albergar un camino por cada par de switches origen-destino, por lo que, cuando seleccionan un camino, bloquean el resto de sus interfaces para que no puedan ser utilizadas por otros caminos disjuntos entre mismo par origen-destino. Este

efecto se ve claramente en el switch *S3*, que tiene seleccionado el primer camino disjunto entre *S1* y *S6* en la Figura 5.11a, y el segundo camino disjunto entre *S1* y *S5* en la Figura 5.11b. En ambos casos, cuando el mensaje de selección de camino llega a *S3* es rechazado, ya que todas sus interfaces están bloqueadas al tener seleccionado un camino previo, por lo que el proceso de selección del tercer camino se ve bloqueado. El resto del proceso es igual que para el modo disjunto en enlaces, los mensajes de selección de camino avanzan hasta alcanzar el switch origen.

En cuanto a la estructura de datos utilizada por el proceso de selección de caminos, ésta mantiene el mismo formato que la estructura del protocolo I-MDP (Figura 5.6b). Cada entrada de la tabla está compuesta por una tripleta que describe de forma inequívoca a cada camino (direcciones MAC de los switches origen y destino e identificador de camino), a la que le acompañan las dos interfaces que utiliza el switch en dicho camino disjunto. Hay que recordar que esta tabla se encuentra en cada uno de los switches de la red y se modifica cuando los switches procesan los mensajes de selección de camino, añadiendo una nueva entrada en la tabla cuando procesan un nuevo camino disjunto. Además, esta tabla contiene solamente información local del switch, por lo que cada switch solo conoce los caminos disjuntos que pasan por él. Para encontrar la interfaz libre de mayor prioridad, los switches buscan en la tabla de exploración la entrada asociada al proceso de selección de caminos en curso, y eligen la primera interfaz de la lista (en la Figura 5.9 se correspondería con la *INTERFAZ A*). Una vez seleccionada, buscan dicha interfaz en la tabla de caminos confirmados, para comprobar que no esté siendo utilizada por otro camino disjunto entre el mismo par de switches origen-destino. Si la interfaz está libre, la seleccionan y añaden una nueva entrada en la tabla de caminos confirmados, en caso contrario, eligen la siguiente interfaz de la tabla de exploración y repiten el proceso. La búsqueda termina cuando se encuentra una interfaz válida, o no quedan interfaces libres en la tabla de exploración. En el modo disjunto en nodos el proceso de búsqueda en las estructuras de información se simplifica. Si la interfaz de la tabla de exploración con mayor prioridad se encuentra en la tabla de caminos confirmados finaliza el proceso de selección de camino, ya que dicho switch tiene un camino disjunto confirmado entre el par de switches origen-destino.

En el ejemplo mostrado en esta sección se han marcado como switches frontera *S1*, *S5* y *S6*, y solamente es *S1* el que realiza el proceso de exploración. En este mismo escenario se podrían lanzar al mismo tiempo otros dos procesos de exploración desde *S5* y *S6*, que obtuviesen caminos disjuntos con los swiches *S1* y *S6*, y *S1* y *S5*, respectivamente. Sin embargo, no se garantiza que los caminos obtenidos por procesos de búsqueda ejecutados en sentidos opuestos sean disjuntos entre sí. Por ejemplo, *S1* origen y *S5* destino obtiene un conjunto de caminos disjuntos, *S5* origen y *S1* destino obtiene otro conjunto de caminos disjuntos, pero la disjuntividad entre ambos conjuntos no está garantizada. También es posible marcar todos los switches como switches frontera, y que la búsqueda de caminos se haga entre un switch origen y el resto de switches que forman la red. Además, gracias a la independencia entre procesos y el sistema de identificación único de camino, en este último

caso también es posible realizar búsquedas simultáneas desde cada uno de los switches frontera hacia el resto de switches frontera.

5.3.3 Mejoras

Durante la fase de diseño se descubrieron pequeños detalles que mermaban el número de caminos disjuntos descubiertos por el protocolo 1S-MDP con respecto a los resultados esperados. Esta sección se centra en describir cada uno de los problemas encontrados y las mejoras introducidas para resolverlos, por ello se va a dividir en dos partes denominadas salto atrás y modificación de prioridades.

Salto atrás

Como se ha visto en la Figura 5.11, durante el proceso de selección de caminos disjuntos en nodos, el proceso de búsqueda solamente obtiene dos caminos disjuntos entre el switch origen y los dos switches destino. Este efecto se debe al bloqueo de interfaces que realiza el switch *S3* por tener un camino disjunto previamente confirmado, lo que impide que dicho switch sea utilizado por otros caminos. El proceso de selección de caminos cuando llega al switch *S3* se detiene porque el switch está utilizado y no puede continuar. La solución que se propone pasa por que el proceso de selección pueda retroceder a switches previos para buscar ramas alternativas que culminen correctamente el proceso de selección del camino. La Figura 5.12, ilustra con detalle el proceso de salto atrás utilizando el escenario de la Figura 5.11b. El switch *S5* genera el tercer paquete de selección de caminos y lo envía al switch *S6*, que a su vez selecciona la interfaz más prioritaria, actualiza el paquete y lo reenvía hacia el switch origen a través de *S3* (paquete marcado con *1)* en la ilustración). El switch *S3* detecta que no tiene interfaces disponibles, por lo que actualiza el paquete y

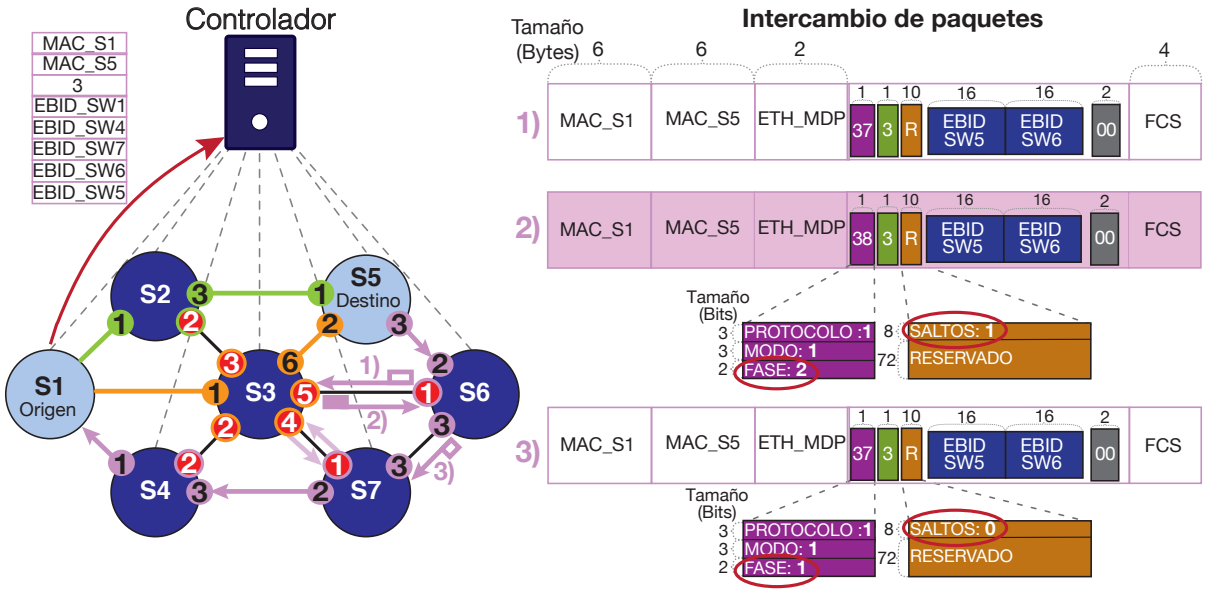


Figura 5.12: Proceso de salto atrás.

lo envía para atrás. Para llevar a cabo este proceso se ha tenido que modificar ligeramente el diseño del mensaje, acondicionando el primer *byte* del campo reservado para anotar el número de saltos que el paquete retrocede. Además, el campo *FASE* ahora puede tener tres valores: 0 para el proceso de exploración, 1 para el proceso de selección de caminos y 2 para el proceso de salto atrás. Por lo tanto, la actualización del mensaje llevada a cabo por el switch *S3* consiste en actualizar el campo *FASE* a 2 y aumentar el número de saltos en una unidad, cuyo resultado es el paquete marcado en la figura como *2)*. El mensaje de salto atrás llega al switch *S6*, quien busca una nueva interfaz libre. En este caso, encuentra la interfaz 3 libre, por lo que vuelve a actualizar el paquete para reenviarlo hacia el switch origen (paquete *3)*). Cuando el paquete llega al switch *S7* ocurre lo mismo, la primera interfaz prioritaria libre es 1, por lo que lo reenvía a través de ella. El mensaje llega a *S3*, lo actualiza y lo manda de vuelta a *S7*, que finalmente encuentra una interfaz libre que permite acabar el proceso de selección del tercer camino. El límite máximo de saltos hacia atrás se alcanza cuando el paquete retrocede hasta el switch destino, dado que el switch destino utiliza todas sus interfaces para iniciar los procesos de selección de caminos disjuntos, por lo que no tiene la capacidad de buscar nuevas interfaces libres. No obstante, el número de máximo de saltos hacia atrás se puede limitar para evitar un retroceso tan grande. Cuando se producen varios saltos hacia detrás en cadena, el último switch (aquel que encuentra una interfaz válida) tiene que eliminar las EBID correspondientes a los saltos atrás que ha dado el paquete, por lo que utiliza el valor del campo *SALTOS* para eliminar las últimas EBID añadidas al paquete. Sin embargo, cuando hay un solo salto hacia detrás no hay que eliminar ninguna EBID del paquete, puesto que el switch que rechaza el paquete en primera instancia no ha añadido su EBID.

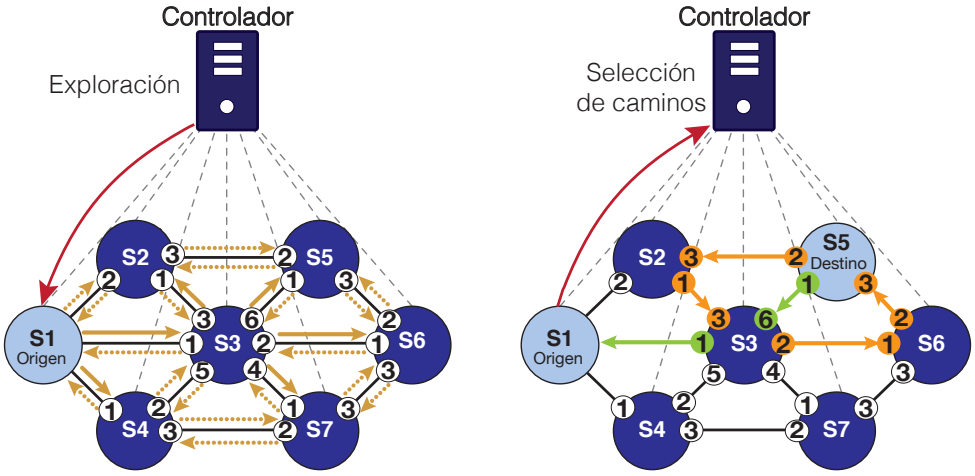
Con este proceso de salto atrás se elimina el bloqueo que sufre la fase de selección de caminos al llegar a un switch que no tiene interfaces disponibles, o que está bloqueado por tener confirmado un camino disjunto previo. Su procedimiento es simple, ya que se basa en enviar el mensaje de selección de camino hacia detrás, siendo los switches que preceden al punto de bloqueo, los responsables de buscar una rama alternativa. En cuanto a las modificaciones que ha sufrido el diseño original, han sido simples: utilizar el primer *byte* del campo reservado del paquete de datos, y añadir una nuevo estado al campo *FASE* del paquete para indicar que el paquete retrocede hacia atrás.

Modificación de prioridades

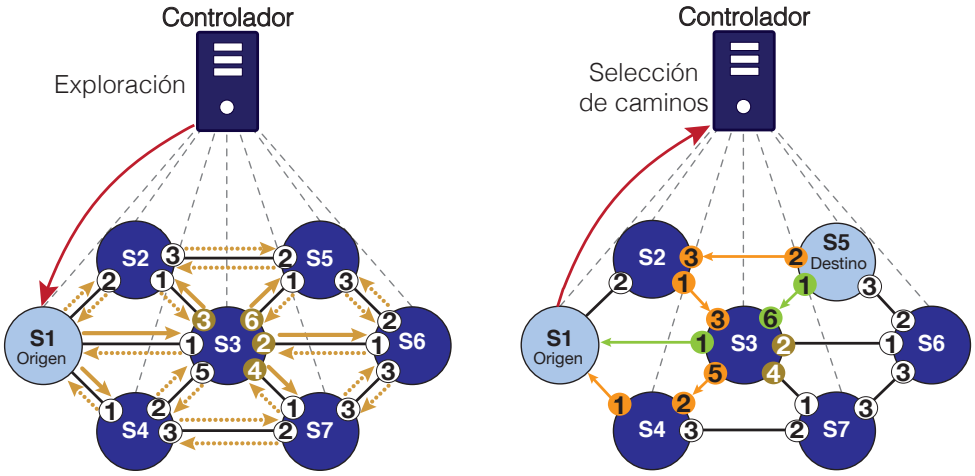
1S-MDP aprovecha la latencia de los paquetes de exploración para fijar niveles de prioridad en las interfaces del switch, ordenándolas en función del instante de llegada del paquete de exploración. Con este sistema, las interfaces que primero reciben los paquetes de exploración son las que tienen mayor prioridad. Sin embargo, este sistema puede presentar contraindicaciones que afectan a la cantidad de caminos descubiertos. La solución pasa por modificar ligeramente el orden de prioridades, disminuyendo la prioridad de aquellas interfaces que presentan conflicto.

Para comprender mejor el problema y la solución que se plantea, se va a representar un escenario de ejemplo con 1S-MDP trabajando en modo disjunto en enlaces. La topología elegida es la misma que en las secciones anteriores, solo que las condiciones de la red han cambiado, por lo que el resultado de la fase de exploración cambia con respecto al ejemplo anterior, dado que las latencias de la red son diferentes. La Figura 5.13a muestra el nuevo escenario de ejemplo con la estructura de búsqueda original, mientras que la Figura 5.13b muestra el mismo escenario aplicando las nuevas políticas de prioridad. En el escenario original (Figura 5.13a), el orden de los puertos en los switches *S2*, *S3* y *S5* ha cambiado con respecto al ejemplo utilizado en los apartados anteriores. Ahora, *S3* y sus enlaces forma un nodo de comunicación extremadamente rápido a través del cual llega la primera copia del paquete de exploración a los switches *S2*, *S5*, *S6* y *S7*. El enlace entre *S3* y *S6* tiene además una latencia mínima, por lo que la copia del mensaje de exploración enviada por *S6* es la segunda en llegar a *S3*. A priori, este es el comportamiento ideal del protocolo y no tiene que provocar anomalías, sin embargo, la fase de selección de camino muestra lo contrario. La selección del primer camino transcurre con normalidad, mientras que en el proceso de selección del segundo se produce un lazo cerrado que parte desde el switch *S5* y termina en el switch *S5*, lo que frustra los procesos de selección del segundo y tercer camino (Figura 5.13a). Cuando el paquete de selección del segundo camino llega al switch *S3*, éste elige la primera interfaz libre con mayor prioridad (la interfaz marcada con un 2) que le conduce hasta el switch *S6* y del switch *S6* a *S5*. Este hecho ocurre porque el diseño original de las técnicas de exploración de red están pensadas para hacer el proceso de exploración en un sentido (de origen a destino) y el de selección de camino en sentido contrario (de destino a origen). Por eso, I-MDP no reenvía el paquete de exploración a través de la interfaz de entrada, para evitar que se produzca un cambio de sentido que origine lazos cerrados. El efecto de lazo cerrado con origen y destino en el mismo switch es el efecto que se produce en 1S-MDP, ya que se efectúan cambios de giro con respecto al sentido lógico del proceso de selección de camino, al escoger un enlace que no se hubiese elegido en condiciones de búsqueda normales. En la literatura existen propuestas que resuelven este problema aplicando técnicas de prohibición o de control de giros [Starobinski et al., 2003, De Pellegrini et al., 2006, Fidler & Einhoff, 2004], por lo que basándose en esta idea se propuso una solución al problema. La solución consta

de dos partes, en primer lugar, durante la fase de exploración se disminuye la prioridad de las interfaces que reciben paquetes de exploración provenientes de un vecino cuya interfaz recibió la primera copia del paquete de exploración. En segundo lugar, una vez que están identificadas las interfaces afectadas, durante la fase de selección de caminos, los switches, al seleccionar las interfaces libres, solamente realizan cambios de giros cuando no quedan interfaces sin penalizar libres. El efecto de esta modificación queda patente en la Figura 5.13b donde se observa que el proceso de exploración ha disminuido la prioridad de ciertas interfaces del switch *S3* (marcadas en rojo). Las interfaces modificadas son aquellas que han recibido un paquete de exploración proveniente de un vecino que recibió la primera copia del paquete de exploración a través de *S3*. Para identificar qué interfaces deben ser penalizadas debe modificarse levemente el paquete de exploración, incluyendo un *bit* en el campo reservado del mismo que se activa cuando un switch reenvía el paquete de exploración a través de la interfaz por la que recibió la copia más rápida del paquete de exploración. De esta forma, los switches, cuando reciben un paquete de exploración, solo tienen que comprobar si el flag está activo para disminuir la prioridad de la interfaz



a) Proceso de búsqueda original.



b) Proceso de búsqueda modificado.

Figura 5.13: Cambio en la política de prioridades.

por la que lo reciben. Una vez que se han caracterizado todas las interfaces, el proceso de selección de caminos cambia su política de selección en el segundo camino, cuando el mensaje llega al switch *S3*, que elige la única interfaz que queda libre sin penalizar (interfaz marcada con un 5). Esta elección conserva el sentido de giro y permite que el proceso de selección del segundo camino se complete. En este último escenario también existiría un proceso de selección para el tercer camino, pero no se ha representado para no complicar la figura. Además, este último tercer camino no llegaría a completarse debido a la morfología del primer y del segundo camino, ya que partiría del switch *S5*, pasaría por *S6* y llegaría hasta *S3*, que elegiría la interfaz penalizada marcada con un 4 debido a que no tiene interfaces sin penalizar libres. Esta elección llevaría al proceso de selección hasta el switch *S7* y después al switch *S4* donde no existiría ninguna salida posible.

El proceso de modificación de la prioridad de las interfaces, junto con la política de cambio de giro, eliminan el problema de la aparición de lazos cerrados con origen y final en el switch destino. Este problema no solo bloquea el proceso de selección en el que aparece, sino que afecta a otros procesos de selección iniciados en el switch destino, lo que se traduce en una merma de los caminos disjuntos descubiertos por el protocolo. La solución, como se ha explicado, es sencilla, basta con añadir un flag al mensaje de exploración que identifica las interfaces a las que disminuir la prioridad, y cambiar la política de selección de interfaces.

5.3.4 Características adicionales

1S-MDP mejora las cualidades de su predecesor al generar más información en la fase de exploración, lo que permite, entre otras cosas, generar, con un solo proceso de exploración, múltiples búsquedas entre un switch origen y múltiples switches destino en paralelo. Gracias al sistema de identificación único de camino, el protocolo también puede descubrir concurrentemente múltiples caminos disjuntos entre un par de nodos, y ejecutar, al mismo tiempo, varios procesos de exploración desde múltiples switches origen, lo que reduce el tiempo de convergencia del protocolo en obtener múltiples caminos disjuntos entre múltiples pares de switches.

Además, el procedimiento de etiquetado por prioridades basado en el instante de recepción de los paquetes de exploración hace que los caminos disjuntos obtenidos entre un par de switches origen-destino estén ordenados por latencias, siendo el primer camino el de menor latencia entre dicho par de switches. Este hecho se produce por el método de selección de interfaces, llevado a cabo por cada switch en la segunda fase, ya que cada switch selecciona la interfaz libre de mayor prioridad o, lo que es lo mismo, la interfaz libre con menor retardo. Este método de selección hace que los caminos se construyan basándose en mínimos locales, por lo que, a excepción del primer camino, los caminos obtenidos no serán necesariamente caminos de mínima latencia, sin embargo, sí que serán caminos cuyo coste en latencia irá incrementándose (el coste del primer camino es menor

que el del segundo, el coste del segundo menor que el del tercero, etc.).

1S-MDP elimina la dependencia de temporizadores, ya que el sistema en su totalidad está orientado a eventos de recepción de paquetes. De esta forma, se elimina el proceso de configuración de los temporizadores, dependiente de un operador de la red, al mismo tiempo que se suprimen los indeseados tiempos de espera entre procesos que incrementan el tiempo de convergencia del protocolo para obtener los caminos disjuntos. El único elemento que puede depender de un sistema basado en temporizadores es el mecanismo de borrado de las tablas de exploración y selección de caminos. Sin embargo, gracias a SDN y su gestión centralizada, se puede delegar dicho mecanismo de borrado al controlador, que bien a través del envío de un mensaje a través del canal de control, bien haciendo uso de tecnologías como P4, puede borrar la información de las tablas de cada uno de los switches.

5.4 Uso de los caminos por parte del controlador

El cometido de I-MDP y 1S-MDP es descubrir caminos disjuntos en redes SDN híbridas, liberando al controlador de la red de su cálculo. Ambos protocolos desplazan el proceso de búsqueda desde el controlador a la capa de infraestructura, y son los switches de la red, que de forma distribuida, obtienen los caminos disjuntos. Sin embargo, la información sobre la composición de los caminos disjuntos sí que es devuelta al controlador, para que los servicios de red ejecutados en el controlador puedan hacer uso de los caminos disjuntos si lo requieren. En las secciones anteriores se ha visto el proceso de búsqueda y el retorno de la información descubierta al controlador, pero no se ha escenificado cómo opera el controlador para hacer uso de los caminos disjuntos.

La Figura 5.14 muestra el proceso que sigue el controlador para hacer uso de los caminos disjuntos. En primer lugar, un servicio de red demanda al controlador el uso de caminos

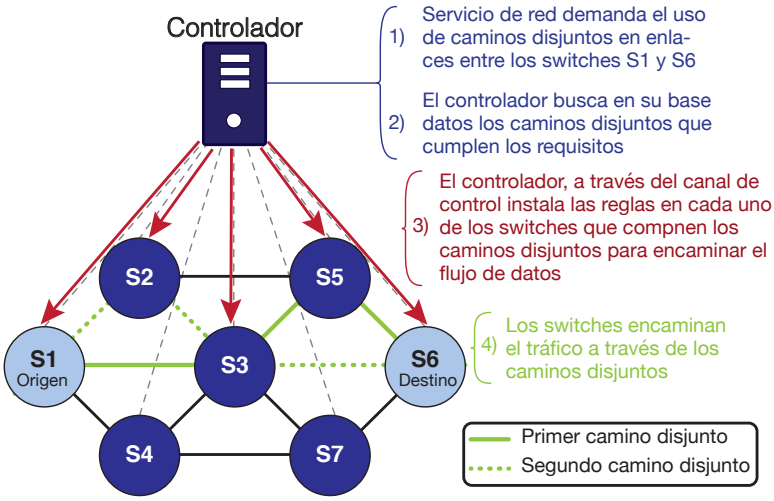


Figura 5.14: Uso de los caminos disjuntos por parte del controlador.

disjuntos en enlaces entre dos switches de la red, por lo que el controlador, bien inicia el proceso de descubrimiento de caminos disjuntos con I-MDP o 1S-MDP y espera a recoger los datos, bien busca en la base de datos los caminos disjuntos si lanzó el proceso de forma proactiva. Tras obtener el conjunto de caminos disjuntos que cumplen con los criterios que demanda el servicio, el controlador obtiene de las EBIDs que componen el camino, los identificadores de los switches, y las interfaces ocupadas en cada switch por los caminos disjuntos. Con esa información, el controlador, a través del canal de control, instala las reglas en cada uno de los switches involucrados que permiten encaminar los flujos de datos acorde a las especificaciones del servicio demandado como, por ejemplo, el envío de un flujo multimedia UDP a través del primer camino disjunto, y el tráfico web a través del segundo camino disjunto. Finalmente, tras instalar las reglas, los switches encaminarán el tráfico acorde a los requisitos impuestos por el servicio sin la intervención del controlador.

5.5 Implementaciones de IMDP y 1S-MDP

El procedimiento seguido para implementar, tanto I-MDP como 1S-MDP, cambia con respecto al de Amaru, dada la duración que supuso dicho proceso. Partiendo de la experiencia de Amaru, se optó por utilizar un simulador de red basado en eventos discretos, y se escogió aquel que ofreciese un modelo de switch SDN más próximo a un switch real. Se optó por utilizar un simulador de eventos discretos debido al ratio calidad de resultados/escalabilidad que ofrecen este tipo de sistemas, ya que es posible simular grandes redes de comunicaciones con un equipamiento hardware limitado, obteniendo resultados próximos a los que se obtendrían en entornos reales. Además, el uso este tipo de simuladores permite analizar en tiempo de ejecución todos los registros del sistema, lo que facilita una depuración a bajo nivel para encontrar errores de código, acelerando los tiempos de desarrollo.

El simulador elegido es ns-3 [Riley & Henderson, 2010], puesto que es un simulador con licencia de código libre orientado a entornos de red, que cuenta con una amplia variedad de protocolos tanto de redes cableadas como de redes inalámbricas y que, además, posee una librería del switch SDN BOFUSS adaptada al simulador. Ns-3 es un simulador de eventos discretos ampliamente utilizado por la comunidad científica que está en constante evolución, incluyendo nuevas funciones, o mejorando las librerías de versiones anteriores. La librería BOFUSS [Chaves et al., 2016] tuvo un peso importante en la toma de decisión para elegir a ns3, ya que anteriormente se trabajó con el switch BOFUSS en la implementación de Amaru, lo que facilita el trabajo.

La librería BOFUSS adaptada a ns-3 introduce, básicamente, dos cambios con respecto a la versión original del switch, que no suponen cambios estructurales, sino más bien sustitución de funciones. Por un lado, se sustituye la función que obtiene el tiempo del sistema operativo por la función que obtiene el tiempo del simulador, adaptando así

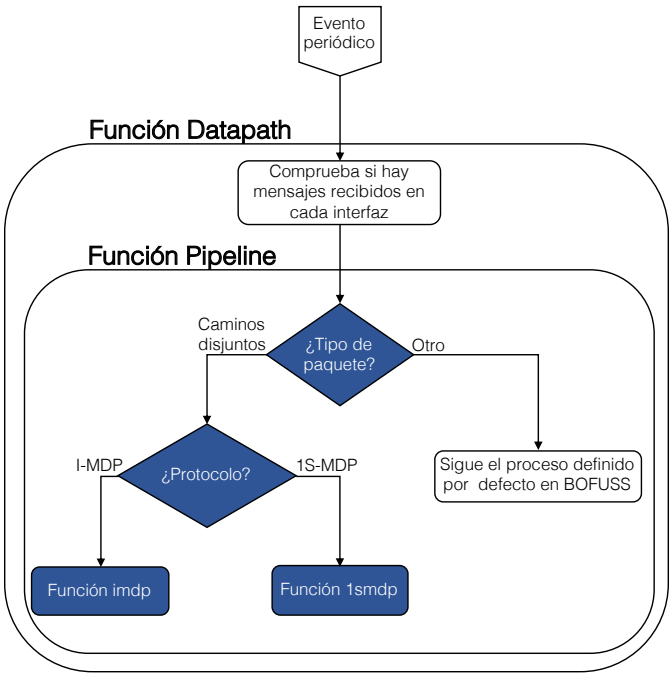


Figura 5.15: Diagrama de flujo general.

toda la temporización del switch al simulador ns3. Por otro lado, se cambia la función que envía los paquetes hacia las interfaces físicas del switch, y que depende del sistema operativo, por las funciones específicas del simulador, redirigiendo así los paquetes al sistema de simulación para que los gestione. Con estos pequeños cambios, se mantiene la estructura original de BOFUSS consiguiendo una temporización y gestión de los paquetes dependientes del simulador.

La Figura 5.15 muestra el diagrama de flujo de la implementación de I-MDP y 1S-MDP en el switch BOFUSS, cuya estructura interna está compuesta, entre otras muchas funciones y estructuras, por dos funciones, la función *datapath* y la función *pipeline*, con las que se va a trabajar. La función *datapath* comprueba periódicamente si hay nuevos paquetes por procesar y, si los hay, los manda a la función *pipeline* para ser procesados. Es en la función *pipeline* donde se llevan a cabo las actuaciones para implementar I-MDP y 1S-MDP, modificando la secuencia de procesado de los paquetes. Las modificaciones introducidas en esta función están marcadas en azul y consisten, básicamente, en comprobar el *Ethertype* del paquete para verificar si pertenece a los protocolos I-MDP o 1S-MDP. Si el paquete no pertenece a los protocolo de búsqueda de caminos disjuntos, entonces el paquete sigue el proceso definido por defecto en BOFUSS, que aplicará las políticas de reenvío definidas por el controlador. Sin embargo, si el paquete pertenece a alguno de los protocolos, éste será redirigido a la función correspondiente para su procesado (*imdp* o *1smdp*). Dado que cada una de estas funciones es independiente y su composición contiene varias fases, se va a dedicar un apartado específico a cada una de ellas para mostrar con mayor nivel de profundidad su estructura interna.

5.5.1 Función *imdp*

La función, representada en la Figura 5.16, analiza, en primer lugar, el campo *FASE* del paquete recibido, identificando la fase del protocolo a la que pertenece el paquete, bien a la fase de exploración, bien a la fase de selección. Cuando el paquete pertenece a la fase de exploración, la función anota tanto las direcciones MAC origen y destino del paquete como la interfaz de entrada del mismo, y comprueba si el paquete recibido se corresponde con la primera copia del proceso de exploración. Si lo es, entonces introduce los datos en la tabla de exploración y difunde dicho paquete a través de todas las interfaces, excepto por la interfaz de entrada y por las interfaces que pertenecen a otros caminos disjuntos entre el mismo par de switches origen-destino, para que el paquete de exploración continúe con el proceso de difusión por la red por los enlaces que no pertenecen a otros caminos disjuntos. En caso contrario, el paquete es rechazado, puesto que el protocolo I-MDP solo realiza el aprendizaje y reenvío con la primera copia del paquete de exploración.

Una vez que ha realizado el proceso de difusión, el switch tiene que comprobar si es el switch frontera destino para iniciar el proceso de selección de caminos. Si es el switch destino, tiene que añadir una nueva entrada en la tabla de caminos confirmados con las direcciones MAC origen y destino del paquete, el identificador de camino y la interfaz de entrada del paquete. En este switch, al tratarse de uno de los extremos del camino, solamente tiene asignada una interfaz para el camino disjunto, ya que el camino no "atraviesa" el switch, sino que llega a él a través de una interfaz. Posteriormente, el switch genera el paquete de selección de caminos incluyendo en él la información de su EBID y lo envía hacia el switch origen a través de la interfaz de entrada del paquete de exploración.

Si el paquete recibido pertenece a la fase de selección, el switch, en primer lugar, anota las direcciones MAC origen y destino del paquete, el identificador de camino y la interfaz de entrada del paquete, y después, comprueba si es el switch extremo del camino. Si dicho switch fuese el extremo (switch origen), el proceso de selección termina, puesto que el paquete de selección de camino ha alcanzado su destino. En este caso, el switch extrae del paquete la información relevante del camino disjunto descubierto (EBIDs, identificador de camino y direcciones MAC del los switches origen y destino) y se la envía al controlador para que gestione los caminos disjuntos descubiertos. En caso contrario, se trata de un switch que está entre los extremos del camino, por lo que tiene que buscar una interfaz que complete el proceso de selección. Para ello, el switch selecciona la interfaz guardada en la tabla de exploración puesto que en este protocolo, en cada switch, por cada proceso de exploración, solo se almacena una interfaz (la que corresponde a la primera copia del paquete de exploración). Cuando ha seleccionado la interfaz de la tabla de exploración, añade una nueva entrada en la tabla de caminos confirmados indicando las interfaces que utiliza el camino disjunto que está en construcción. Por último, incluye su EBID en el paquete de selección de caminos y lo reenvía reenvía hacia el switch origen a través de la

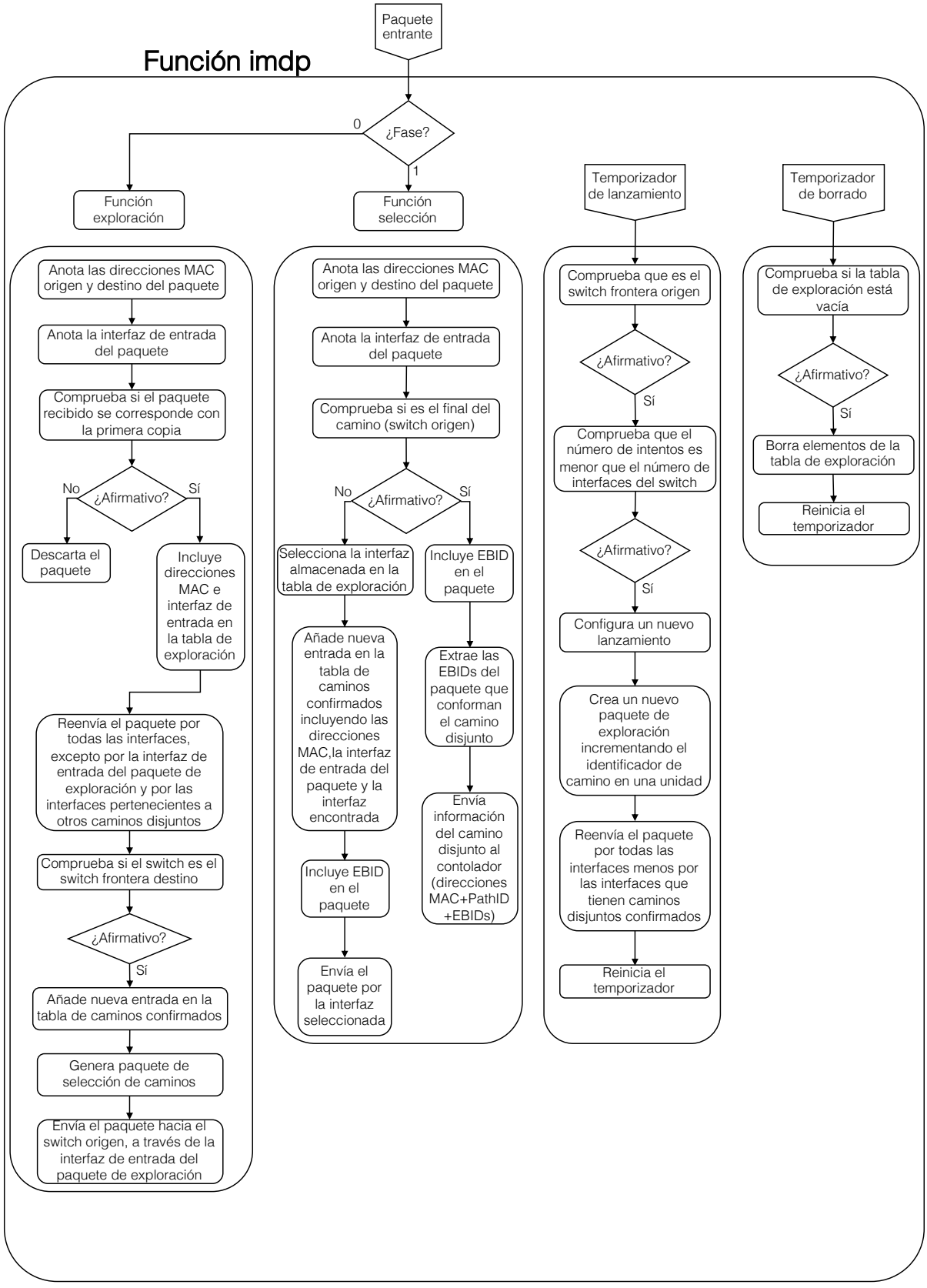


Figura 5.16: Diagrama de flujo de la función *imdp*.

interfaz seleccionada en la tabla de exploración.

En este protocolo también existen dos procesos periódicos para ejecutar los procesos de exploración secuenciales entre caminos, y borrar la información de procesos de exploración previos. El tiempo de ambos temporizadores lo establece, bien el controlador, bien el administrador de la red en el arranque del sistema, y su inicio se fija al iniciar un nueva búsqueda de caminos disjuntos entre un par de switches origen-destino. El temporizador de lanzamiento comprueba que el número de procesos de exploración ejecutados para ese par de switches origen-destino es menor que el numero de interfaces del switch, esta comprobación es necesaria para ejecutar tantos procesos de exploración como interfaces tiene el switch. Si la comprobación es positiva ejecuta un nuevo proceso de exploración y reinicia el temporizador para que vuelva a saltar. Con respecto al temporizador de borrado, borra los elementos de la tabla de exploración siempre que esta no esté vacía.

5.5.2 Función *1smdp*

La función *1smdp*, representada en la Figura 5.17, al igual que la función *imdp*, analiza el campo *FASE* del paquete recibido para redirigirlo a la función correspondiente (exploración, selección o salto atrás). La función de exploración procesa el paquete de exploración anotando, en primer lugar, las direcciones MAC origen y destino del paquete, la interfaz por la que lo reciben y el instante temporal en el que se recibe para establecer el nivel de prioridad en la tabla de exploración. Además, como la función incorpora las mejoras introducidas en la Sección 5.3.3, la función comprueba el flag del campo reservado para decrementar el nivel de prioridad si es necesario. Tras almacenar la información en la tabla de exploración, el switch continúa el proceso de difusión reenviando la primera copia del paquete de exploración a través de todas sus interfaces, lo que garantiza un proceso de difusión controlado libre de bucles. Además, en cada difusión, el switch debe activar el flag reservado para decrementar la prioridad de una interfaz cuando el paquete es reenviado por la interfaz que recibió la copia más rápida del paquete de exploración.

Una vez que se han realizado las tareas correspondientes a la fase de exploración, el switch comprueba si es un switch frontera destino, verificando que la dirección MAC destino del paquete de exploración se corresponde con su dirección MAC, con la dirección de grupo multicast que tiene asignada, o con la dirección de broadcast. Si es un switch destino, tiene que iniciar el proceso de selección de caminos, creando una nueva entrada en la tabla de caminos confirmados, generando un paquete de selección de caminos en el que incluye su EBID, y enviándolo hacia el switch origen a través de la interfaz por la que recibió el paquete de exploración.

La función de selección analiza todos los paquetes de selección que llegan a cada switch. En primer lugar, la función obtiene del paquete las direcciones MAC origen y destino, anota la interfaz de entrada del paquete y comprueba si es el switch origen. En caso de serlo, calcula su EBID y extrae del paquete las EBIDs que componen el camino disjunto

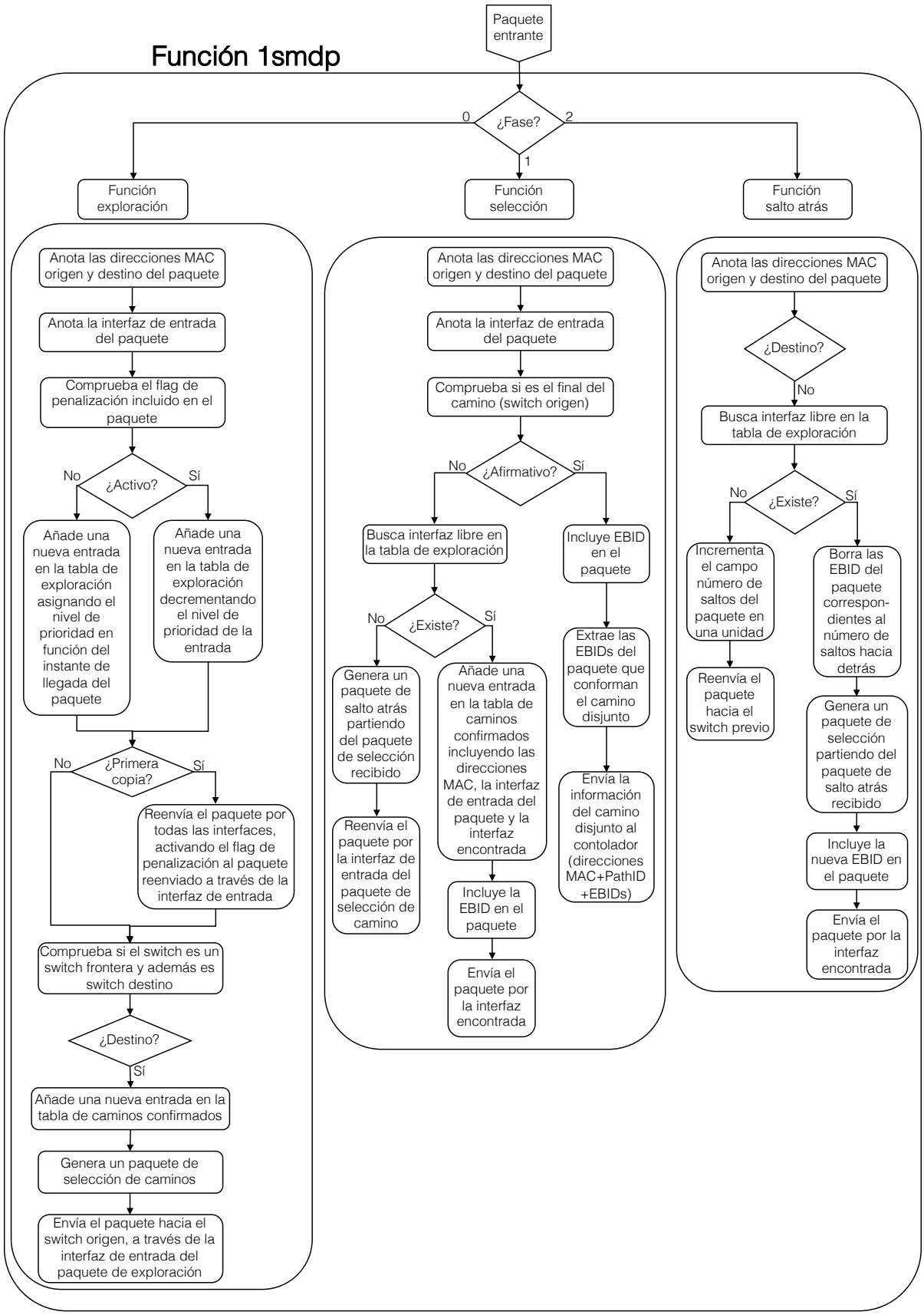


Figura 5.17: Diagrama de flujo de la función *1smdp*.

para enviarle al controlador la composición del camino descubierto. En caso contrario, busca la interfaz libre de mayor prioridad por la que reenviar el paquete de selección de camino hacia el switch origen. Si la encuentra, añade una nueva entrada en la tabla de caminos confirmados, calcula su EBID, la incluye en el paquete y lo reenvía hacia el switch origen a través de la interfaz encontrada. Si no encuentra una interfaz libre, bien porque no quedan interfaces en el modo disjunto en enlaces, bien porque el modo nodo tiene un camino previamente confirmado, genera un paquete de salto atrás y lo envía a su predecesor en la cadena, con el objetivo de que el switch previo encuentre una rama alternativa.

Por último, la función de salto atrás, comprueba que no es el switch destino para buscar la interfaz de mayor prioridad libre en la tabla de exploración. Si existe dicha interfaz, actualiza el paquete borrando las EBIDs de los saltos que ha retrocedido dicho paquete, calcula su nueva EBID, e introduce toda esta información en un nuevo paquete de selección de camino que es reenviado por la interfaz libre encontrada. Sin embargo, si la búsqueda resulta infructuosa, el switch envía el paquete de salto atrás al switch que le precede, incrementando en una unidad el campo del paquete que guarda el número de saltos.

5.6 Evaluación

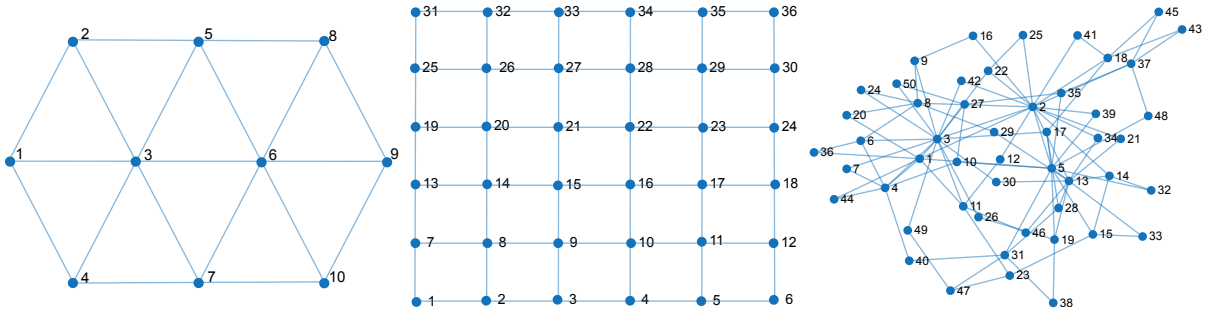
Esta sección se centra en evaluar los protocolos de descubrimiento de caminos, I-MDP y 1S-MDP, midiendo el número de caminos disjuntos descubiertos y el tiempo que tarda cada propuesta en obtenerlos (tiempo de convergencia). Estas dos métricas ofrecen un estándar de calidad, ya que permiten ver la ratio cantidad de caminos obtenidos versus tiempo invertido de forma gráfica, que servirá para mostrar el comportamiento de cada propuesta. Además, esta sección también analiza como afectan al rendimiento de 1S-MDP las mejoras introducidas.

Además, este apartado también valora los beneficios que aportan las redes SDN híbridas en las tareas de búsqueda de caminos disjuntos frente a soluciones SDN. En el capítulo dedicado al estado del arte se vio que el comportamiento habitual en redes SDN consiste en calcular los caminos disjuntos de forma centralizada en el controlador, partiendo de un grafo de la red, cuya obtención depende de protocolos de descubrimiento topológico. Además, muchas de las propuestas emplean el algoritmo de Dijkstra para calcular los caminos disjuntos, bien inspirándose en él, bien utilizándolo junto con algoritmos heurísticos o técnicas de optimización. Dado que el algoritmo de Dijkstra se emplea como algoritmo base en muchas soluciones SDN, tanto en soluciones que obtienen caminos disjuntos en enlaces como en soluciones que obtienen caminos disjuntos en nodos, se va a utilizar su versión original como propuesta alternativa a los sistemas híbridos, ya que es un buen punto de referencia sobre el que sacar conclusiones.

El proceso que sigue el algoritmo de Dijkstra para obtener los caminos disjuntos consiste en calcular el camino de menor coste entre dos switches de la red, eliminar del grafo dicho camino y volver a aplicar el algoritmo de Dijkstra sobre el nuevo grafo. Al eliminar el camino obtenido del grafo se garantiza que búsquedas posteriores obtengan caminos disjuntos con respecto a los caminos obtenidos en pasos previos. Este método de cálculo emplea un método recursivo de búsqueda que finaliza cuando se descubren todos los caminos posibles entre el switch origen y el switch destino. Por lo tanto, el cálculo de caminos disjuntos con el algoritmo de Dijkstra emula la búsqueda de caminos disjuntos que sigue el controlador en una red SDN. Además, se sitúa al algoritmo de Dijkstra en una ventaja competitiva, ya que solo se tiene en cuenta el cálculo de los caminos, sin incluir el tiempo necesario para descubrir la topología, requisito imprescindible para poder calcular los caminos disjuntos.

El escenario sobre el que se construye la evaluación cuenta con 3 modelos topológicos en los que se varía el tamaño de la red y el grado de conexión entre vecinos, con el objetivo de analizar el comportamiento de los protocolos en varios escenarios y evaluar su escalabilidad. La primera topología sobre la que se evalúan los protocolos comparte estructura con la topología de ejemplo de este capítulo, pero añade tres switches más, lo que proporciona una topología de tamaño contenido y estructura regular que la hace idónea para estudiar el comportamiento inicial de los protocolos y corregir posibles errores. Más adelante, se someten los protocolos a un mayor nivel de estrés, al ejecutarlos sobre topologías regulares (mallas cuadradas bidimensionales), en las que se incrementa el tamaño de la red para ver como escalan las propuestas en redes de mayor dimensión. Por último, se utilizan modelos aleatorios en los que se varía tanto la conectividad entre vecinos como el tamaño de la red para analizar tanto la escalabilidad de los protocolos como la cantidad de caminos disjuntos obtenidos en redes no regulares. Este último tipo de topologías, al igual que en Amaru, se generan utilizando el simulador BRITE con los modelos de conexión Barabási-Albert y Waxman, utilizando además un modelo de distribución espacial *Heavy-Tailed* que ubica los switches en la superficie de forma asimétrica, creando zonas más pobladas de switches y otras menos pobladas. La Figura 5.18 muestra un subconjunto de las topologías utilizadas en la evaluación, representado a partir de los datos empíricos que utiliza ns3 para construir los escenarios. En concreto, la Figura 5.18a recoge la topología de ejemplo, la Figura 5.18b muestra una malla bidimensional de 6x6 cuyo tamaño de red alcanza los 36 nodos, y la Figura 5.18c una topología aleatoria con modelo de conexiones Barabási-Albert de 50 nodos de tamaño y grado de conectividad cuatro.

En todas las topologías los enlaces son bidireccionales y cuentan con un ancho de banda de 1 Gbps, mientras que el retardo de propagación de los mismos varía. En las topologías regulares este retardo se fija en cada enlace siguiendo una función de distribución uniforme entre 0 y 1.5ms, mientras que en las topologías aleatorias el retardo se fija en función de la distancia entre los switches, cuya superficie está calculada para que el retardo de



a) Topología de ejemplo. b) Malla bidimensional. c) Topología aleatoria.

Figura 5.18: Topologías utilizadas en la evaluación.

propagación máximo sea de 1.5ms (la diagonal de un rectángulo de 440km x 88km). Este rango de valores no se ha elegido al azar, sino que tiene una explicación, expuesta a continuación. Para no introducir más ruido al sistema y que la búsqueda de caminos no se vea afectada por elementos externos, los experimentos se realizan sin carga de tráfico. Sin embargo, los protocolos bajo estudio basan su búsqueda en la latencia de la red, por lo que hay que introducir algún sistema que varíe las condiciones temporales de la red para poder calcular valores medios e intervalos de confianza, ya que si la latencia de la red permanece invariante, los resultados obtenidos en cada topología para distintas ejecuciones serán iguales. Por lo tanto, se va a utilizar el tiempo de propagación como variable sobre la que efectuar el modelado de la latencia de la red, ya que es una variable sencilla de cambiar en la plataforma de simulación. Dado que la latencia de un paquete para llegar desde un switch A a un switch B viene determinada por la suma de los tiempos de espera en cola, transmisión y propagación, y que el tiempo de propagación sobre redes SDN en entornos de centros de datos o redes Metropolitan Area Network (MAN) que utilizan enlaces de fibra óptica es prácticamente despreciable, el retardo de propagación modelará un escenario de red con carga de tráfico, donde dicho retardo se corresponde con el tiempo adicional de espera en cola provocado por el tráfico de datos. De ahí que el límite máximo del tiempo de propagación sea 1.5ms, puesto que es el tiempo de espera en cola para una cola de 125 paquetes de 1500 *bytes* ($(1500\ bytes \cdot 8\ bits/byte \cdot 125\ pkt)/1\ Gbps = 1{,}5ms$). Esos 125 paquetes tampoco se han escogido al azar, sino que se han tenido en cuenta datos empíricos de un switch SDN como es el Pica8, que tiene un búfer compartido para todos sus puertos y niveles de prioridad por puerto. Este switch con 125 paquetes por cola de prioridad y puerto tiene un nivel de congestión medio-bajo suficiente para ralentizar los flujos de datos sin llegar a colapsar el sistema [Van Bemten et al., 2019].

En cuanto a las plataformas utilizadas, para los protocolos I-MDP y 1S-MDP se utiliza el simulador de eventos discretos ns-3 con la librería BOFUSS adaptada para dicho simulador [Chaves et al., 2016], mientras que el cálculo de caminos disjuntos con el algoritmo de Dijkstra se ejecuta en la herramienta de cálculo matemático Matlab. Se escogió Matlab porque solo se quiere tener en cuenta el proceso de cómputo que se lleva a cabo en el controlador, excluyendo otras variables externas como el cálculo del grafo,

por lo que cualquier plataforma en la que se pueda ejecutar el algoritmo de Dijkstra es válida. Además, Matlab cuenta con la librería de Wang [Wang, 2004], que implementa de forma óptima el algoritmo de Dijkstra y, al mimo tiempo, incluye numerosas herramientas de cálculo y representación matemática, motivos que decantaron la balanza para utilizar Matlab.

Todos los experimentos se han ejecutado sobre la misma plataforma hardware, un ordenador con sistema operativo Linux que cuenta con un procesador Intel(R) Core(TM) i7-8700K CPU, 32 GB RAM y disco duro de estado sólido de 256GB. Además, cada experimento se repite 10 veces y se calculan los valores medios de los 10 lanzamientos y los intervalos de confianza al 95 %. La Tabla 5.1 resume el proceso de evaluación llevado a cabo con las características principales de cada prueba.

5.6.1 Resultados

Este apartado se encarga de mostrar el número de caminos disjuntos descubiertos por I-MDP, 1S-MDP y Dijkstra en los tres modelos topológicos anteriormente expuestos, así como el tiempo que ha tardado cada protocolo en obtener dichos caminos caminos disjuntos (tiempo de convergencia). La prueba consiste en calcular todos los posibles caminos disjuntos entre todos los switches de la red. Además, el protocolo 1S-MDP incluye sus dos mejoras (salto atrás y modificación de prioridades en los puertos). En las pruebas realizadas, tanto I-MDP como 1S-MDP, inician de forma simultánea todos los procesos de búsqueda de caminos disjuntos, dado que su diseño lo permite, lo que reduce el tiempo de convergencia, mientras que en el algoritmo de Dijkstra este proceso se tiene que hacer de forma secuencial, ejecutando un único proceso de búsqueda entre un par de switches cada vez, al tratarse de un sistema de cómputo centralizado. Las pruebas comienzan mostrando los resultados de la topología más sencilla (topología de ejemplo), para después presentar los resultados de las topologías más complejas (mallas bidimensionales y aleatorias).

La Tabla 5.2 muestra los valores medios y el intervalo de confianza al 95 % del número de caminos disjuntos en nodos y en enlaces descubiertos por cada protocolo, y su tiempo de convergencia, en la topología de ejemplo. A la vista de los resultados, es 1S-MDP el que, de lejos, obtiene un tiempo de convergencia menor, sin embargo, descubre menos caminos que sus competidores. Estos resultados no son casuales, sino que se deben al proceso de exploración único que realiza cada switch para descubrir múltiples caminos disjuntos con múltiples destinos. Al lanzar un solo proceso de exploración por cada switch origen se

Tabla 5.1: Resumen del escenario de evaluación.

Topología	**Velocidad enlaces (Gbps)**	**Retardo enlaces (ms)**	**Tamaño de la red (nodos)**	**Conectividad**	**Número de ejecuciones**	**Medidas**
Ejemplo	1	0 a 1.5	10	3.8	10	Número de caminos disjuntos y tiempo de convergencia
Malla bidimensional	1	0 a 1.5	4 a 100	2 a 3.6		
Aleatorias	1	0 a 1.5	10 a 100	2, 4, 6		

Tabla 5.2: Resultados de la topología de ejemplo.

			Dijkstra	I-MDP	1S-MDP
Disjunto en nodos	Tiempo de convergencia (s)	Media	0.2299	13.97	**0.0058**
		95 % i.c	0.0875	1.27	0.0009
	Número de caminos descubiertos	Media	**232.60**	**232.60**	230.20
		95 % i.c	9.69	9.69	7.51
Disjunto en enlaces	Tiempo de convergencia (s)	Media	0.1087	18.52	**0.0105**
		95 % i.c	0.040	0.0022	0.0018
	Número de caminos descubiertos	Media	**281.8**	**281.8**	266.40
		95 % i.c	1.32	1.32	7.52

reduce el tiempo invertido en explorar la red (ya que solo se explora una vez, frente las múltiples exploraciones que realizan sus competidores), que, junto con la concurrencia de procesos de exploración ejecutados desde distintos switches origen, hace que el tiempo de convergencia se reduzca drásticamente. En cuanto al número de caminos descubiertos, en modo disjunto en nodos, 1S-MDP obtiene el 94 % de los caminos con respecto a sus competidores, y en modo enlaces el 99 %. Esta diferencia también se debe al proceso único de exploración, ya que 1S-MDP solamente recoge información topológica una vez, mientras que I-MDP y Dijkstra realizan un proceso de exploración/análisis por cada camino descubierto entre un par de switches. Estos procesos iterativos de exploración/análisis de la red describen con mayor nivel de detalle el estado de dicha red en cada momento, a costa de extender el tiempo de convergencia, por lo que son capaces de descubrir más caminos disjuntos. El efecto visible es que en modo nodo, al tener una política más restrictiva, los múltiples procesos de exploración extraen mejor la información topológica en I-MDP y Dijkstra, viéndose penalizado 1S-MDP en cuanto al número de caminos descubiertos. Con respecto al tiempo de convergencia, llaman la atención los valores, demasiado elevados, que obtiene I-MDP, efecto asociado directamente al uso de temporizadores para ejecutar los procesos de exploración secuenciales entre caminos disjuntos de un mismo par de switches origen-destino.

La pequeña diferencia existente en relación al número de caminos descubiertos por 1S-MDP con respecto a las otras dos propuestas no es preocupante, ya que, normalmente, las aplicaciones no necesitan todos los caminos disjuntos disponibles, sino solamente un subconjunto de estos. Por ejemplo, en [Raiciu et al., 2011] se afirma que con cuatro caminos es suficiente para obtener el 80 % de la velocidad máxima posible del flujo en topologías de redes de centros de datos, o [Marina & Das, 2006, Nasipuri et al., 2001] que establecen una horquilla entre dos y tres caminos alternativos para encaminar el tráfico de forma óptima en redes inalámbricas. Sin embargo, sí que es importante obtener los caminos en el menor tiempo posible para reducir el tiempo de espera a la aplicación que demanda el servicio de caminos múltiples disjuntos, faceta en la que destaca 1S-MDP.

La Figura 5.19 muestra los resultados obtenidos por las tres propuestas en topologías malladas, cuyo tamaño de red oscila entre los cuatro y los cien switches. La mitad izquierda de la figura muestra el tiempo de convergencia, y la mitad derecha el número de caminos

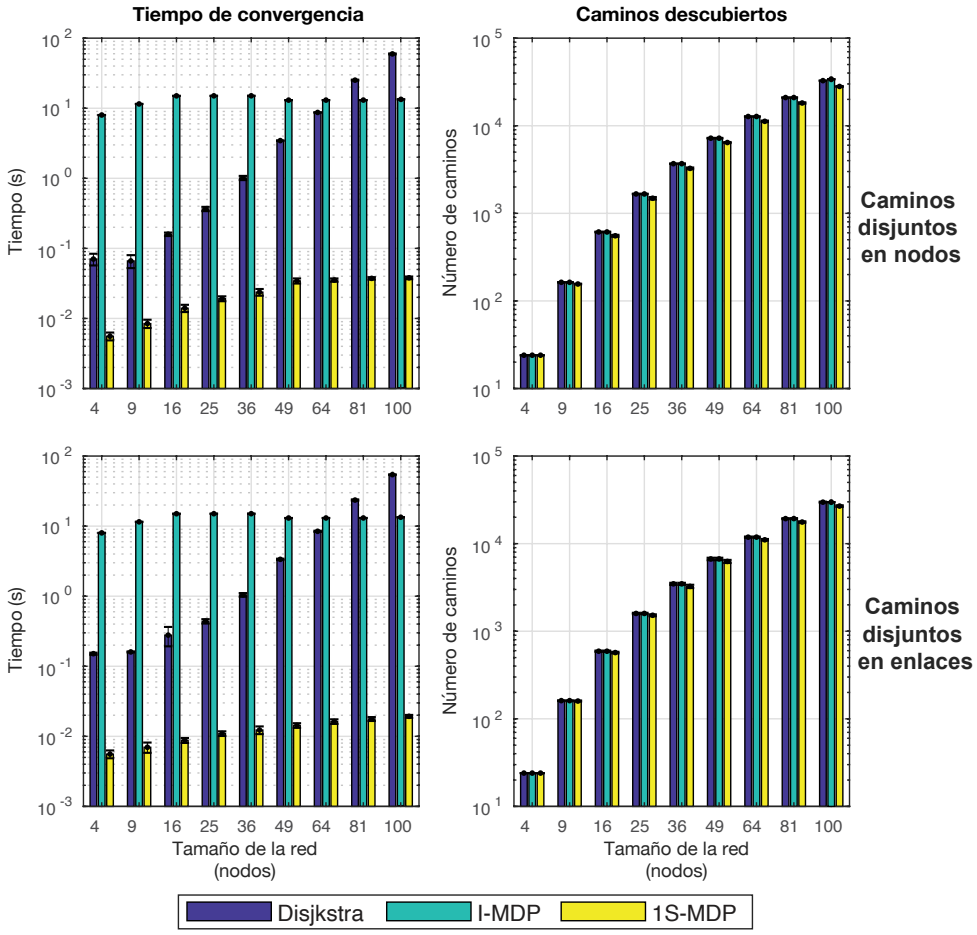
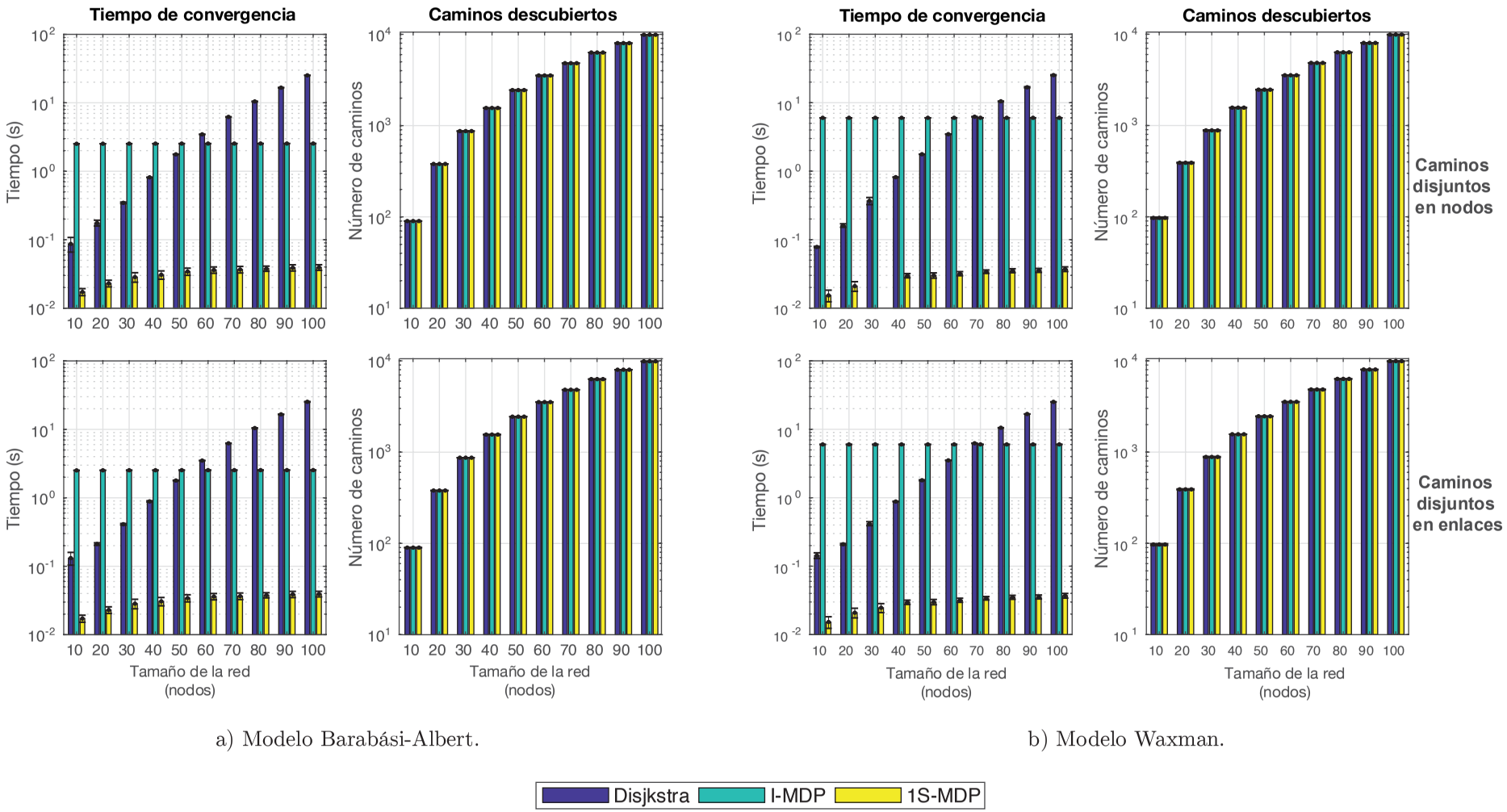


Figura 5.19: Resultados para la topología mallada.

descubiertos, mientras que la fila superior recoge los resultados del modo disjunto en nodos y la fila inferior del modo disjunto en enlaces. Los resultados obtenidos muestran la misma tendencia que en la topología de ejemplo, 1S-MDP obtiene un 12 % menos de caminos disjuntos en nodos que sus competidores y un 7 % menos de caminos disjuntos en enlaces en la topología de cien switches. Sin embargo, el tiempo de convergencia se reduce entre dos y tres órdenes de magnitud, lo que compensa el pequeño déficit de caminos descubiertos. Además, si se observa la evolución de las gráficas, según se incrementa el tamaño de la red, se ve como el tiempo de convergencia de Dijkstra crece exponencialmente. Este efecto se debe a que el algoritmo de Dijkstra tiene que calcular de manera centralizada, en un único punto, todos los caminos disjuntos de la topología, mientras que 1S-MDP distribuye el proceso de búsqueda entre todos los dispositivos, por lo que la cantidad de cómputo por dispositivo se mantiene constante. Aún así, se observa un ligero crecimiento en el tiempo de convergencia de 1S-MDP debido a que el proceso de exploración necesita más tiempo para explorar al completo redes de mayor tamaño. En cuanto al tiempo de convergencia de I-MDP, éste permanece constante con valores superiores a sus competidores, debido de nuevo a su diseño basado en temporizadores.

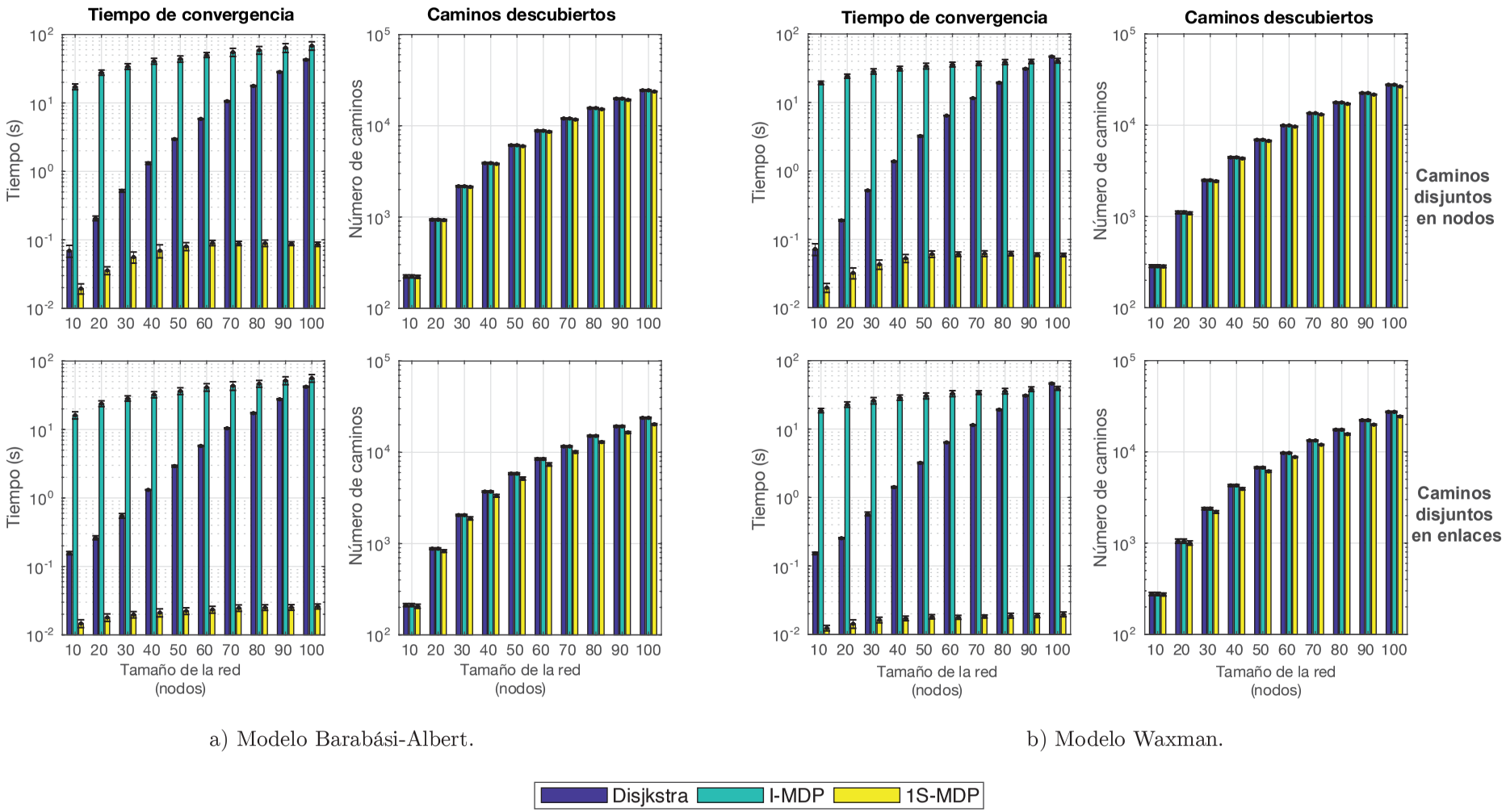
Por último, las Figuras 5.20, 5.21 y 5.22 muestran los resultados obtenidos en topologías aleatorias con grados de conectividad dos, cuatro y seis, respectivamente, para los modelos



a) Modelo Barabási-Albert.

b) Modelo Waxman.

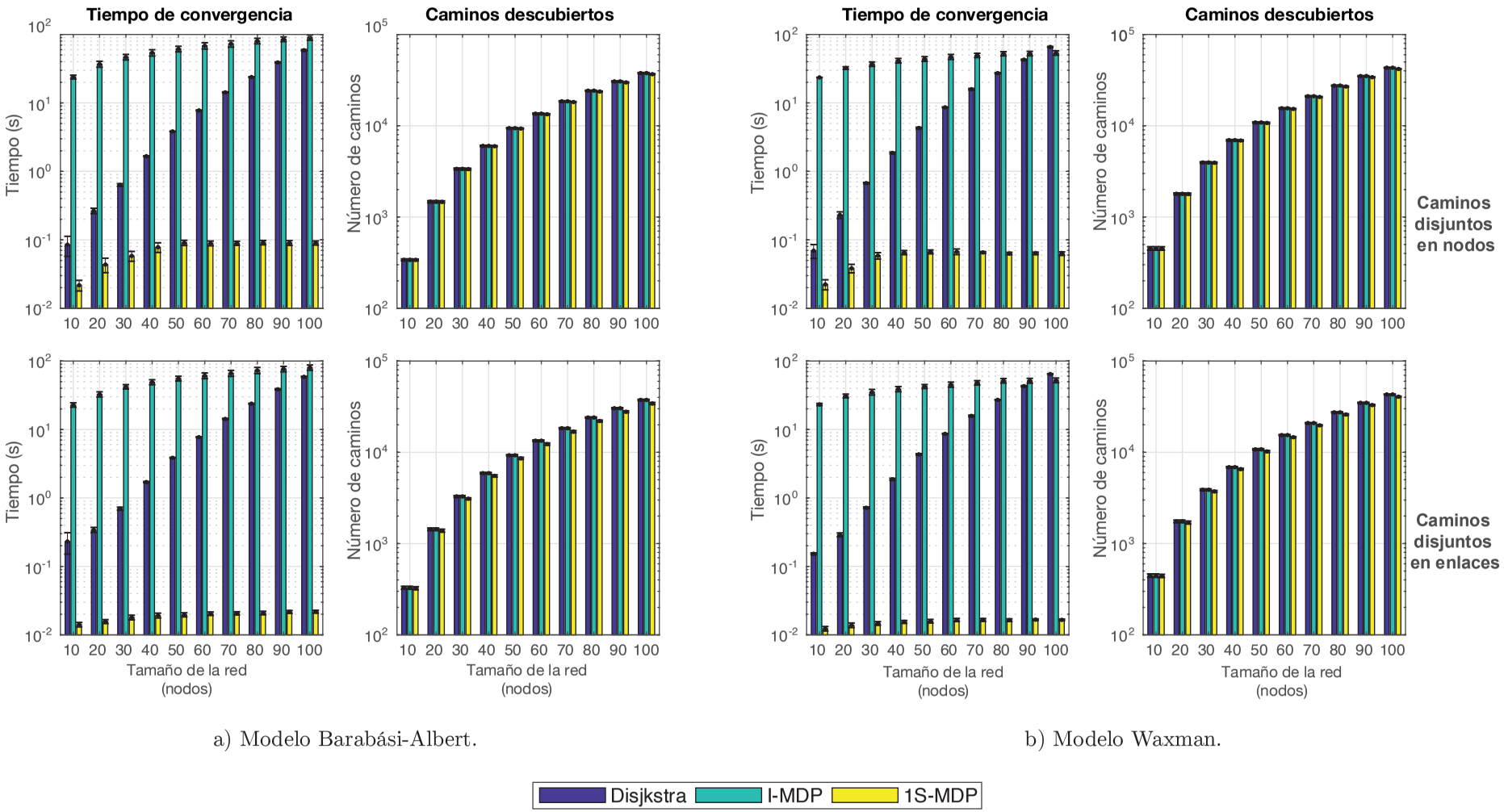
Figura 5.20: Resultados para la topologías aleatorias con una conectividad de grado dos.



a) Modelo Barabási-Albert.

b) Modelo Waxman.

Figura 5.21: Resultados para la topologías aleatorias con una conectividad de grado cuatro.



a) Modelo Barabási-Albert.

b) Modelo Waxman.

Figura 5.22: Resultados para la topologías aleatorias con una conectividad de grado seis.

de conexión Barabási-Albert y Waxman. Todas las figuras comparten la misma estructura que la figura anterior, mostrando, en la fila superior, el tiempo de convergencia y los caminos descubiertos del modo caminos disjuntos en nodos, mientras que la fila inferior, recoge los resultados del modo caminos disjuntos en enlaces. Independientemente del modelo de conexión, del grado de conectividad y del tamaño de la red, 1S-MDP vuelve a ser la propuesta que obtiene los caminos en menor tiempo, reduciendo en más de tres órdenes de magnitud el tiempo de convergencia en topologías con alta conectividad y tamaño de red, y manteniendo la mejora entre dos y tres órdenes de magnitud en el resto de topologías. Además, 1S-MDP mantiene el porcentaje de caminos descubiertos con respecto a sus competidores, mostrando oscilaciones entre el 2 y el 12 % en función de los tres parámetros anteriormente mencionados (modelo de conexión, grado de conectividad y tamaño de la red). En relación al tiempo de convergencia, el comportamiento se vuelve a repetir, manteniéndose constante para I-MDP y creciendo exponencialmente en Dijkstra. De nuevo 1S-MDP vuelve a reducir considerablemente el tiempo de convergencia con respecto a los otros dos protocolos.

Hay un caso que llama especialmente la atención, las topologías con un grado dos de conectividad y un tamaño de red superior a los 70 switches. En este escenario, el tiempo de convergencia del algoritmo de Dijkstra supera al de I-MDP, debido a que el grado tan bajo de conectividad reduce los efectos adversos que provoca el uso de temporizadores. En las topologías con un grado dos de conectividad, en media, I-MDP ejecutará dos procesos de exploración para descubrir los caminos disjuntos entre un par de switches origen-destino, por lo que el temporizador saltará solamente una vez (para iniciar el proceso de exploración del segundo camino). Por lo tanto, el tiempo de espera muerto entre procesos se reduce, y por ende, el tiempo de convergencia. Al verse reducido el tiempo de convergencia de I-MDP, rápidamente el algoritmo de Dijkstra lo supera, debido a la problemática que supone el cálculo de caminos en un único punto centralizado.

Verificación de las mejoras introducidas

Este apartado dentro del capítulo de resultados pretende mostrar la eficacia de las mejoras introducidas en el protocolo 1S-MDP: el salto atrás y la modificación de prioridades. Ambas características están habilitadas en los resultados mostrados durante esta sección, sin embargo, ahora se deshabilitan para mostrar el efecto que tienen en los resultados. Para este nuevo apartado no se han utilizado todas las topologías, sino que se ha escogido un subconjunto suficientemente representativo que muestra el efecto positivo de estas mejoras. Dada la estructura irregular que presentan las topologías aleatorias, así como la heterogeneidad entre las conexiones de los switches (un switch puede tener dos vecinos, y otro switch 10), este tipo de topologías es ideal para presentar el problema, ya que existen más casos de switches poco conectivos que en las topologías regulares, incrementando los puntos de actuación en los que será necesario el uso del salto atrás para finalizar satisfactoriamente la construcción de los caminos disjuntos. Además, este tipo de topologías también es apropiado para mostrar el efecto positivo de la modificación de prioridades, ya que el uso de interfaces no prioritarias será más habitual en topologías irregulares que en topologías regulares, puesto que las topologías regulares, gracias a su estructura, disponen de muchas más interfaces prioritarias disponibles antes de tener que utilizar las penalizadas. Por todo ello, se ha escogido un subconjunto de las topologías aleatorias con 20, 40, 60 y 80 nodos, y tres grados de conectividad (dos, cuatro y seis vecinos por nodo) que permiten evaluar el efecto de estas mejoras.

En primer lugar, se va a evaluar el comportamiento del salto atrás, midiendo la mejoría

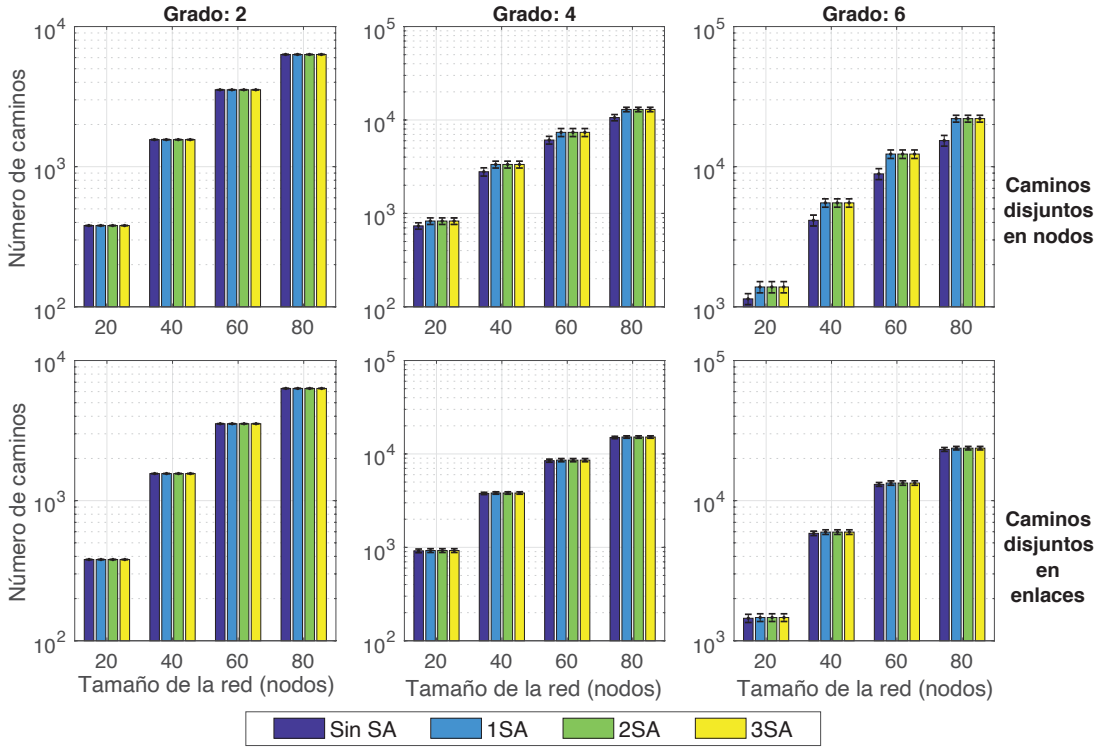


Figura 5.23: Caminos obtenidos, modelo Barabási-Albert.

que se obtiene al activarlo, y cuál es el número máximo de saltos recomendado, a partir del cual, no se incrementa el número de caminos disjuntos obtenidos. La Figura 5.23 muestra el número de caminos descubiertos por 1S-MDP en la topología aleatoria con el modelo Barabási-Albert para distintos tamaños de red, con tres grados distintos de conectividad, cuando se limita el número de saltos atrás. Las gráficas muestran los caminos que se obtienen cuando no se permite el salto hacia detrás (sin SA), y cuando se limita el número de saltos a uno, dos, y tres saltos, 1SA, 2SA, y 3SA, respectivamente. En las topologías con grado de conectividad bajo el salto atrás no es efectivo, puesto que la conectividad entre vecinos es muy baja, por lo que no se encuentran vías alternativas aunque se retroceda. Este comportamiento se repite en modo caminos disjuntos en enlaces, independientemente del grado de conectividad y el tamaño de la red debido, a que en modo caminos disjunto en enlaces, un mismo switch puede albergar varios caminos disjuntos, por lo que rara vez un switch no tiene interfaces libre disponibles. Sin embargo, el salto atrás es verdaderamente efectivo en el modo caminos disjuntos en nodos, dónde claramente se observa el efecto positivo de implementar esta medida. Para garantizar que los caminos sean disjuntos en nodos, cada switch, debe pertenecer en exclusiva a uno de los caminos disjuntos existentes entre un par de switches origen-destino, por lo que durante el proceso de selección de caminos a menudo se llega a un switch utilizado por otro camino disjunto, siendo necesario retroceder a switches anteriores para encontrar una rama alternativa.

Para ver el efecto que tiene limitar el número de saltos hacia atrás se utiliza la Figura 5.24. En ella se representa, en forma de porcentaje, el incremento de caminos

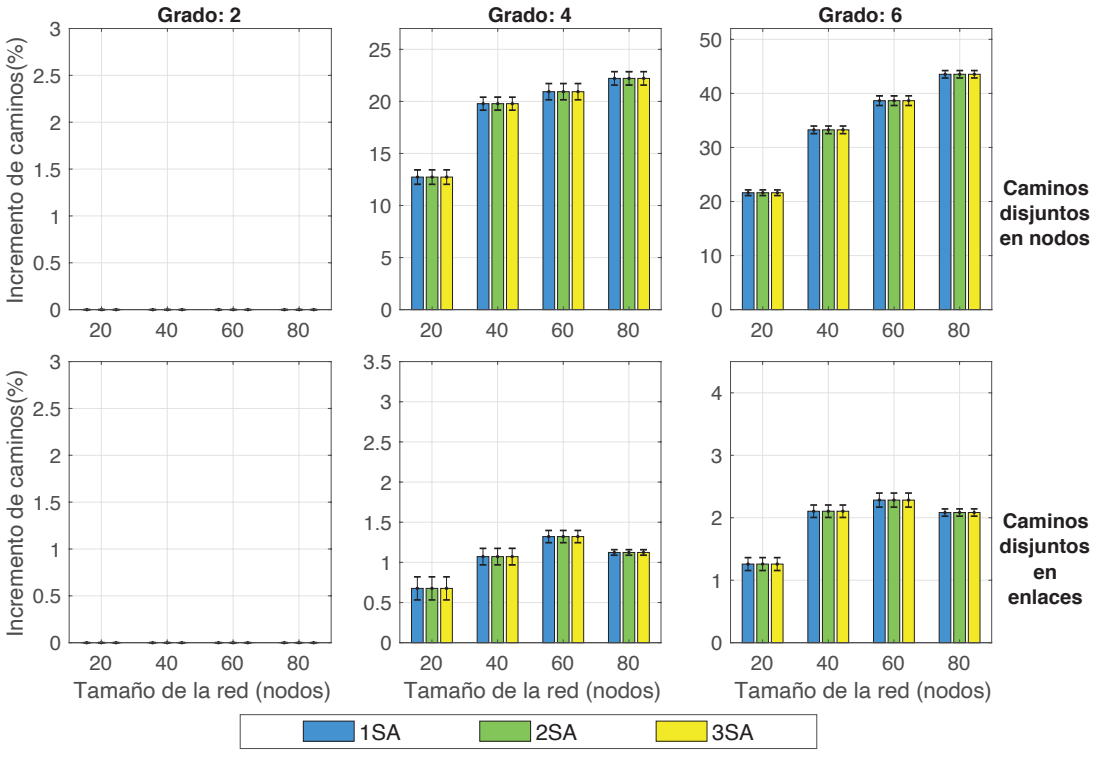


Figura 5.24: Incremento de caminos obtenidos, modelo Barabási-Albert.

obtenidos con respecto a tener deshabilitado el salto atrás. La tendencia es la misma que la descrita anteriormente, el salto es poco efectivo en modo caminos disjuntos en enlaces pero realmente necesario en modo caminos disjuntos en nodos, consiguiendo hasta un 45 % más de caminos disjuntos en nodos. Además, se observa que con un único salto hacia atrás es suficiente para alcanzar dichas ratios, ya que no se incrementa el número de caminos obtenido conforme se incrementa el número de saltos hacia detrás posibles.

Por último, las Figuras 5.25 y 5.26 muestran los mismos resultados para el modelo de Waxman, donde se vuelve a comprobar la efectividad del salto atrás en el modo disjunto en nodos, y que no es necesario dar más de un salto atrás para alcanzar el ratio máximo de caminos descubiertos.

En segundo lugar, se muestra el efecto positivo que tiene activar el sistema de modificación de prioridades durante la fase de exploración, ilustrando, de forma gráfica, la mejora que obtiene 1S-MDP, en términos de caminos disjuntos descubiertos, al activarlo. Las Figuras 5.27 y 5.28 recogen el número de caminos descubiertos por 1S-MDP en el subconjunto de topologías seleccionadas para los modelos de conectividad Barabási-Albert y Waxman, respectivamente. Ambas figuras muestran el número de caminos descubiertos por 1S-MDP en dos modalidades: la versión original del protocolo, y la versión mejorada, con el sistema de modificación de prioridades activado. Además, para ver con mayor nivel de detalle las mejoras obtenidas, las figuras incluyen, en forma de porcentaje, el incremento de caminos que obtiene la versión con el sistema de modificación de prioridades activado con respecto a la versión original. Al igual que las figuras previas de este apartado, la

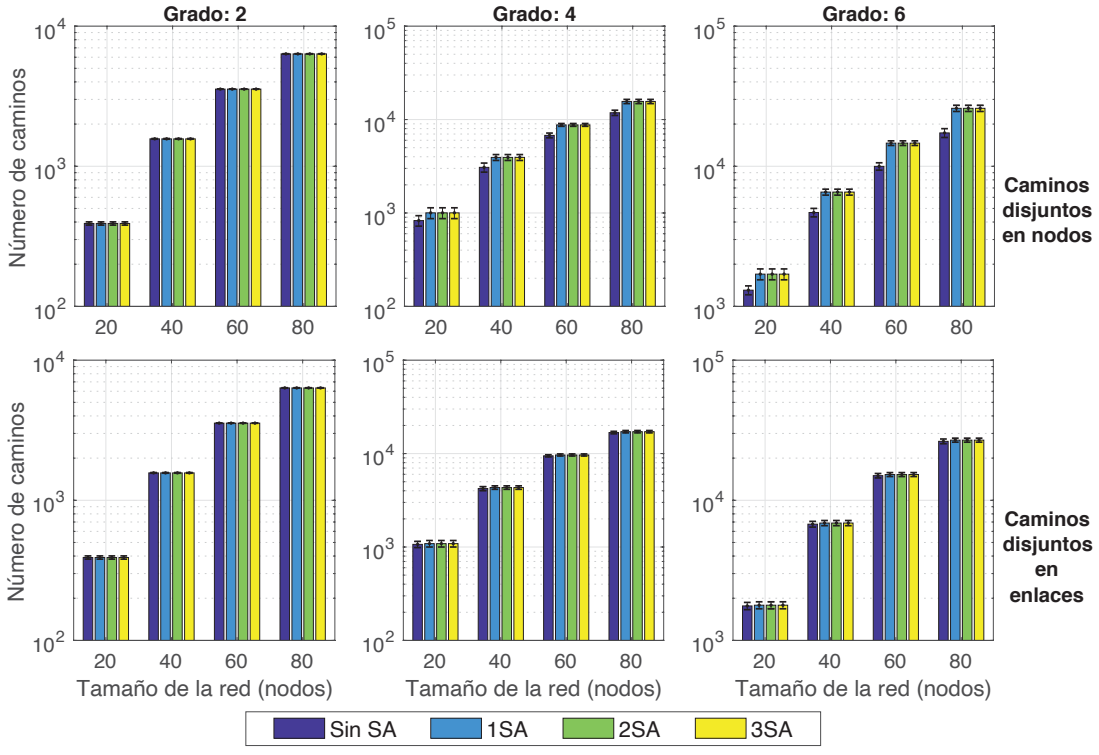


Figura 5.25: Caminos obtenidos, modelo Waxman.

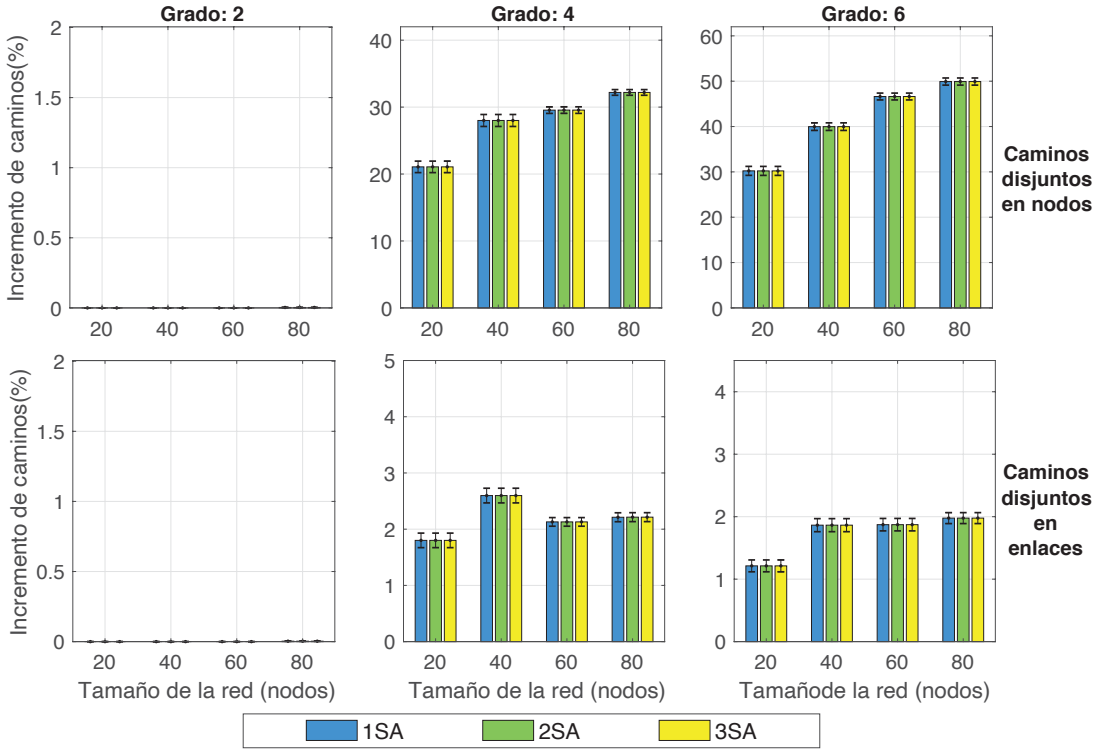


Figura 5.26: Incremento de caminos obtenidos, modelo Waxman.

disposición interna de los gráficos se estructura en dos filas y tres columnas, mostrando en la fila superior los resultados para el modo disjunto en enlaces, y la fila inferior para el modo disjunto en nodos. En cuanto a la disposición por columnas, cada una de ellas

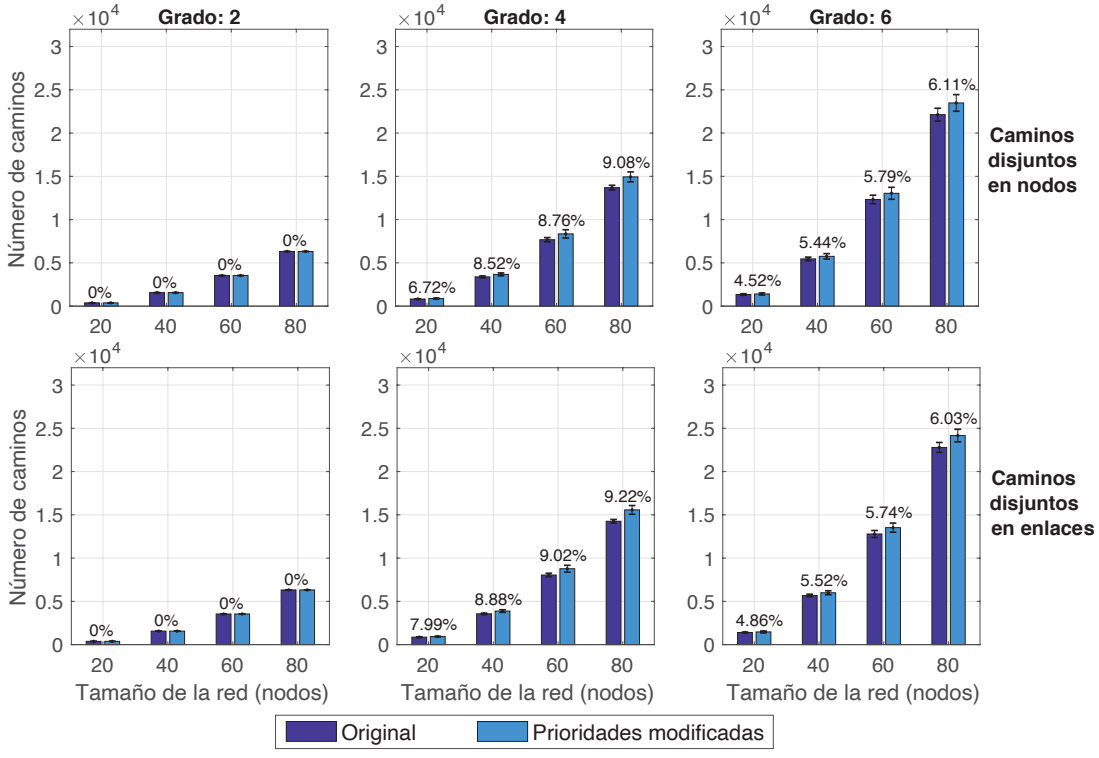


Figura 5.27: Incremento de caminos obtenidos por modificación de prioridades, modelo Barabási-Albert.

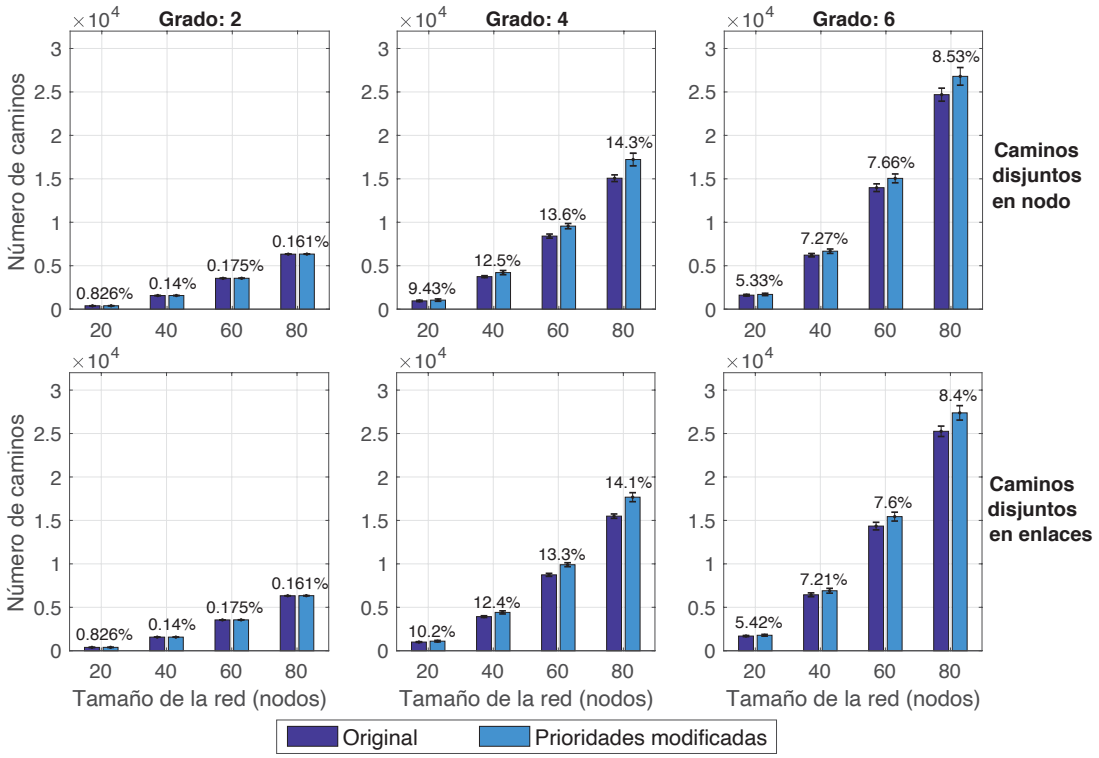


Figura 5.28: Incremento de caminos obtenidos por modificación de prioridades, modelo Waxman.

muestra los resultados para un grado de conectividad (dos, cuatro y seis vecinos por nodo, respectivamente).

En vista de los resultados, el sistema de modificación de prioridades apenas tiene efecto en las topologías con un grado de conectividad bajo (grado dos), mientras que en las de grado cuatro y seis la mejora en el número de caminos descubiertos es notable, incrementando hasta en un 14 % el número de caminos obtenidos. Asimismo, también se observa un incremento del porcentaje de caminos disjuntos descubiertos, con respecto a la versión original, conforme aumenta el tamaño de la red, exceptuando las topologías de grado dos en las que apenas se aprecia mejoría. Sin embargo, en el subconjunto de topologías en las que, con diferencia, se obtiene un mayor rendimiento, es en las topologías con grado de conectividad cuatro, cuyas cotas de mejora casi doblan a las de las topologías de grado seis. Todas estas conclusiones no son casualidad, sino que están relacionadas con el mismo fenómeno, el número de interfaces que requieren modificaciones de prioridad. Los switches muy conectivos son dispositivos clave, ya que a través de ellos llega la primera copia del paquete de exploración a otros switches vecinos, por lo que reciben paquetes de exploración de vuelta con el flag de modificación de prioridades activado. Por lo tanto, en topologías de baja conectividad (grado dos), apenas existen este tipo de dispositivos debido, a la baja conectividad de la red, ya que en media, cada switch tiene dos vecinos. Sin embargo, en topologías de grado cuatro y seis, el número de vecinos por switch aumenta, por lo que se incrementa el número de switches muy conectivos en los que hay que aplicar la modificación de prioridades. En topologías de grado cuatro este suceso es relevante, ya

que existe una conectividad suficiente como para tener que utilizar a menudo interfaces con prioridad modificada para finalizar correctamente la construcción de los caminos. No obstante, este suceso no se ve tan magnificado en topologías con grado seis, ya que cada switch tiene suficientes vecinos como para no tener que utilizar tan a menudo interfaces con la prioridad modificada, por lo que el efecto visible es que el rendimiento es menor que en topologías de grado cuatro.

Por lo tanto, ambos sistemas (salto atrás y modificación de prioridades) son útiles y complementarios para incrementar el ratio de caminos descubiertos por 1S-MDP. Si bien el salto atrás, tiene mayor efecto en el modo caminos disjuntos en nodos, el sistema de modificación de prioridades afianza la mejora en ambos modos de funcionamiento. Además, se ha visto que ambas mejoras son realmente útiles a partir de un grado de conectividad cuatro, por lo que ambos sistemas resultan efectivos en un conjunto amplio de topologías, haciendo necesaria su integración en 1S-MDP.

Capítulo 6

MDPAlg

Este capítulo recoge la tercera, y última, aportación de la Tesis, en la que se produce un cambio de paradigma, pasando de un enfoque distribuido o híbrido, a un enfoque completamente centralizado. Este cambio de contexto se produce debido a la imposibilidad de desplegar redes híbridas en escenarios concretos, como por ejemplo, redes SDN con equipamiento recientemente adquirido que deber ser sustituido al completo por ser incompatible para albergar switches híbridos. Para estos casos no queda más remedio que desplegar un servicio de búsqueda de caminos disjuntos que trabaje de forma centralizada en el controlador. No obstante, se vio que este enfoque centralizado puede ser también interesante para abordar el problema del multicamino en otras disciplinas, abriendo así el abanico de aplicaciones a las que puede ser destinado el algoritmo de búsqueda de múltiples caminos disjuntos. El algoritmo de Dijkstra es un buen ejemplo de ello, ya que su aplicación abarca campos tan dispares como el cálculo de caminos en la aplicación Google Maps [Lanning et al., 2014], la obtención de rutas para robots [Wang et al., 2011] o la segmentación de imágenes médicas [Hong, 2012], para un análisis más rápido. A raíz de la aplicabilidad del algoritmo de Dijkstra y de los resultados recogidos en el capítulo Capítulo 5 en cuanto a la mejora del tiempo de convergencia de 1S-MDP con respecto al algoritmo de Dijkstra, se proyectó la idea de transformar el protocolo 1S-MDP en un algoritmo que trabaje de forma centralizada, cuyo resultado es el algoritmo descrito en este capítulo, MDPAlg.

La idea es aprovechar la metodología utilizada por el proceso de exploración único de 1S-MDP, adaptándola a un sistema centralizado que trabaja sobre un grafo topológico y, aprovechar así, los beneficios de 1S-MDP en cuanto al tiempo de convergencia consumido. Sin embargo, los procesos concurrentes de selección de caminos llevados a cabo por los switches en 1S-MDP no pueden ser adaptados, puesto que el entorno centralizado se basa en la ejecución de un proceso monolítico sobre un único hilo que no admite la ejecución de procesos en paralelo, por lo que el tiempo de convergencia obtenido por MDPAlg será sensiblemente superior al consumido por 1S-MDP.

En este capítulo la nomenclatura cambia ligeramente debido, precisamente, al cambio

de paradigma. Al trabajar en un entorno centralizado sobre un grafo topológico, la red no estará compuesta por switches, sino por nodos, los cuales están conectados entre si a través de un conjunto de enlaces. El método de obtención del grafo es un proceso independiente al algoritmo, quien hace uso del grafo una vez ha sido calculado por otro protocolo/algoritmo. Además, se hablará de complejidad computacional, término que contabiliza el consumo de recursos temporales y/o físicos que necesita el algoritmo para ser ejecutado, y que normalmente está relacionado con el número de nodos y enlaces que componen la red.

El capítulo describe en primera instancia el algoritmo MDPAlg, mostrando el proceso que sigue para realizar la búsqueda de caminos disjuntos. A continuación, en la segunda sección se realiza un análisis teórico de la complejidad computacional del protocolo, comparándola con otros algoritmos encontrados en la literatura para valorar la calidad de la propuesta. Finalmente, se realiza una evaluación de la propuesta que valide todo el desarrollo llevado a cabo a lo largo del capítulo.

6.1 Descripción del algoritmo

MDPAlg se concibe como una versión centralizada del algoritmo 1S-MDP debido a sus buenos resultados en el número de caminos descubiertos y en el tiempo de convergencia obtenido. Al igual que 1S-MDP, MDPAlg cuenta con un proceso de búsqueda dividido en dos fases, siendo la primera fase la encargada de recopilar información topológica, y la segunda la responsable de seleccionar los caminos. El procedimiento seguido es similar, solo que ahora la metodología se adapta para trabajar sobre una representación de la red en forma de grafo, por lo que no existe un intercambio de mensajes entre vecinos para recopilar la información topológica o para seleccionar los caminos, sino que toda esa información se consigue mediante la ejecución de operaciones matemáticas que operan sobre el grafo. MDPAlg también es capaz de obtener, con un único proceso de análisis de los costes del grafo (anteriormente conocido como proceso de exploración), múltiples caminos disjuntos entre un nodo origen y el resto de nodos que componen la red. Asimismo, se utilizará la terminología "nodo frontera" para identificar los nodos que componen los extremos del camino (nodos origen y destino de los caminos), al igual que se hacía en 1S-MDP.

En cuanto al tipo de grafo sobre el que trabaja MDPAlg, éste debe ser un grafo ponderado y no dirigido, si se quiere replicar la metodología de búsqueda de 1S-MDP, ya que el protocolo necesita enlaces *full-dúplex* para llevar a cabo el proceso de búsqueda en dos fases. Sin embargo, MDPAlg puede trabajar también sobre grafos ponderados y dirigidos ya que el algoritmo puede comprobar fácilmente la direccionalidad de los enlaces y hacer un análisis de la red en función de esta direccionalidad. Ahora bien, trabajar con grafos dirigidos presenta un inconveniente para este algoritmo en cuanto a los caminos disjuntos devueltos, ya que, dependiendo del grado de direccionalidad del grafo, el

conjunto de caminos disjuntos proporcionado será, total o parcialmente, unidireccional. No obstante, este tipo de caminos disjuntos unidireccionales pueden ser útiles en aplicaciones concretas, como por ejemplo el diseño de circuitos de sentido único en hospitales con rutas libres de COVID-19 que no se pueden solapar con rutas susceptibles al contacto por COVID-19. En ese tipo de escenario trabajar con grafos dirigidos es ideal, ya que se garantiza la independencia entre las rutas al mismo tiempo que se establecen sentidos únicos de circulación para preservar la distancia de seguridad interpersonal.

Para conocer con mayor nivel de detalle como se ha llevado el todo el proceso de adaptación, esta sección incluye un apartado específico para cada una de las dos fases, describiendo minuciosamente el proceso de búsqueda de caminos disjuntos de MDPAlg.

6.1.1 Primera fase: Análisis de costes

Existe una relación directa entre la fase de exploración de 1S-MDP y la fase de análisis de costes de MDPAlg. Ambas tienen como objetivo explorar la red y ver el coste que tiene viajar desde un punto origen hasta cualquier punto destino. El proceso de exploración realizado por 1S-MDP que, mediante una difusión de mensajes, obtiene información de la latencia acumulada para ir desde un switch origen a cualquier otro punto de la red, se transforma en un proceso de cálculo matemático que obtiene el coste acumulado para ir desde un nodo origen a cualquier otro punto del grafo. Durante el proceso de transformación, se analizaron en detalle los requisitos de MDPAlg, sus características, y las cualidades que debía reunir, y se vio que la fase de análisis de costes de MDPAlg guarda una estrecha similitud con el algoritmo de Dijkstra. Este algoritmo realiza un análisis de costes sobre el grafo topológico para obtener un vector de predecesores con coste acumulado mínimo desde el nodo origen, a través del cual se puede construir fácilmente el árbol de mínimo coste que comunica al nodo origen con el resto de los nodos del grafo. MDPAlg realiza el mismo proceso de análisis, solo que en vez de guardar únicamente información relativa al árbol de mínimo coste, además, almacena información sobre los enlaces entre ramas (*cross-links*), obteniendo así una caracterización completa del grafo con la que posteriormente se calculan los caminos disjuntos. Por lo tanto, dado que existe una estrecha similitud en los análisis de costes que realizan ambos algoritmos, MDPAlg va a basar su diseño en el algoritmo de Dijkstra, sustentándose en que es un trabajo consolidado a lo largo de los años que cuenta con una implementación optimizada. No obstante, se ha de remarcar el hecho de que el algoritmo Dijkstra con un análisis de costes únicamente obtiene un camino entre un nodo origen y un nodo destino, mientras que la información adicional que recoge MDPAlg le permite obtener con un único proceso de análisis múltiples caminos disjuntos entre el nodo origen y el resto de nodos del grafo.

Para comprender mejor todo el proceso y mostrar las diferencias existentes en el análisis de costes que realizan el algoritmo de Dijkstra y MDPAlg, se proporciona el pseudocódigo de ambas propuestas, los Algoritmos 6.1 y 6.2, respectivamente, ilustrándose ambos

procesos de forma gráfica con ayuda de la Figura 6.1. Además, tanto el pseudocódigo del algoritmo de Dijkstra como el de MDPAlg utilizan como referencia la codificación de Dijkstra disponible en [Cormen et al., 2001]. En cuanto a la descripción gráfica, la Figura 6.1a muestra el resultado de aplicar Dijkstra, y la Figura 6.1b el resultado realizar el análisis de costes de MDPAlg. Ambas figuras están dividas verticalmente en dos partes, situando en la parte superior el grafo topológico, y en la inferior las estructuras de datos que devuelven los algoritmos. El algoritmo de Dijkstra devuelve dos vectores, C y P, que contienen el coste (mínimo) para ir desde el nodo origen hasta cualquier otro nodo, y el nodo padre (predecesor en el árbol de expansión creado), a través del cual se consigue el coste mínimo. Partiendo de dicha información se puede construir de manera sencilla el árbol de mínimo coste que parte del nodo origen y alcanza al resto de nodos de la red, ya que los vectores proporcionan el coste mínimo para llegar a cada nodo y el nodo padre el nodo a través del que se llega. Sin embargo, la fase de análisis de MDPAlg devuelve solamente una única matriz, denominada Matriz de Costes (MC), que contiene el coste acumulado para viajar desde el nodo origen hasta el resto de los nodos incluyendo tanto el camino de mínimo coste como los *cross-links*. Esta matriz recoge información del conjunto de enlaces que forman el grafo y, además, los caracteriza con respecto a su coste acumulado desde el nodo origen, lo que simplifica el proceso de construcción de caminos de la segunda fase.

La función que describe el algoritmo de Dijkstra (Algoritmo 6.1), recibe como parámetros de entrada el grafo topológico y el nodo origen, con el fin de proporcionar al algoritmo información sobre el conjunto de nodos que forman la topología, las conexiones entre ellos y el coste asociado a cada conexión, y el nodo origen desde el que se inicia el análisis de costes. La función, en primer lugar, inicializa las variables S, Q, P y C (líneas 2 - 4), que almacenan los nodos visitados, los no visitados, el nodo padre de cada nodo en el árbol de mínimo coste, y el coste que tiene viajar desde el nodo origen hasta el resto de los nodos del grafo a través del árbol de mínimo coste, respectivamente. Aquí es importante mencionar que la intersección entre los vectores S y Q da como resultado un conjunto vacío, y su unión, el conjunto de nodos que forman la topología, puesto que el algoritmo evalúa cada uno de los nodos una única vez. La función *Initialize_single_source* asigna un coste inicial de valor 0 al nodo origen y un coste inicial con valor infinito al resto de los nodos, para asegurar que el algoritmo de Dijkstra comienza el análisis del grafo desde el nodo origen, ya que evalúa los nodos visitándolos en un orden de coste creciente. Después, incluye en el vector Q el conjunto de nodos que forman el grafo topológico y, mientras existan nodos sin visitar ($Q \neq \emptyset$), el algoritmo extrae de Q el nodo que presenta menor coste, lo añade al vector S (líneas 5 -8), y evalúa a sus vecinos aplicándoles la función Relax (R) descrita en [Cormen et al., 2001] (línea 9). Esta función calcula el coste que tiene viajar desde el nodo origen s hasta el nodo v, vecino de u, atravesando u, y actualiza, tanto el coste del nodo v como su nodo padre, si v no se ha visitado con anterioridad y se reduce el coste que tenia previamente almacenado. Los nodos visitados se excluyen de este

cálculo, porque una reevaluación posterior implica necesariamente un incremento de coste, ya que el nuevo coste se obtiene a través de un nodo intermedio extraído posteriormente de Q. Este nodo intermedio, al extraerse más tarde de Q, obligatoriamente tiene un coste superior, ya que la política de extracción de Q sigue un orden de coste incremental, por lo que una reevaluación de un nodo previamente visitado a través de un nodo intermedio extraído posteriormente de Q, siempre obtiene un coste superior El resultado de aplicar el algoritmo de Dijkstra sobre la topología de ejemplo se recoge en la Figura 6.1a, que muestra en su parte superior el grafo topológico, el coste de los enlaces, el árbol de mínimo coste, el padre de cada nodo en dicho árbol, y el coste para ir desde el nodo origen hasta cada nodo a través del árbol de mínimo coste; mientas que, en la parte inferior, se muestran las estructuras de datos que devuelve el algoritmo de Dijkstra, el vector de nodos padre y el vector de costes, a través de los cuales es sencillo construir el árbol de mínimo coste.

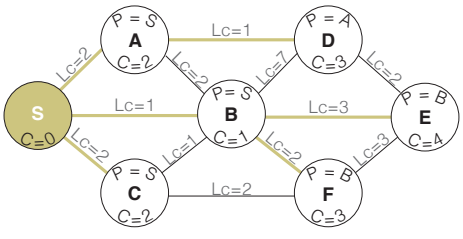
La función de análisis de costes de MDPAlg descrita en el Algoritmo 6.2 recibe los

Algoritmo 6.1 Algoritmo de Dijkstra

```
1:  function DijkstraSAlgorithm(G, s)
2:      Initialize(S, P)
3:      C = Initialize_single_source(G, s)
4:      Q = Get nodes from G
5:      while Q ≠ ∅ do
6:          u = extract_min(Q)
7:          Insert u in S
8:          for each vertex v neighbour of u do
9:              R { if S ∩ v = ∅ then
                    w = L_c from u to v
                    if C(u) + w < C(v) then
                      C(v) = C(u) + w
                      P(v) = u
10:     return C, P
```

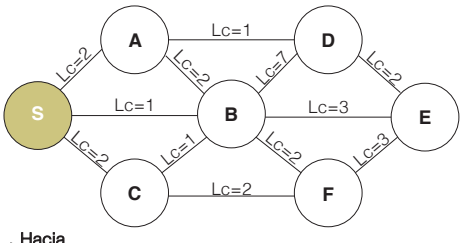
Algoritmo 6.2 MDPAlg: análisis de costes

```
1:  function AnalisisOfCosts(G, s)
2:
3:      MC = Initialize_single_source(G, s)
4:      Q = Get nodes from G
5:      while Q ≠ ∅ do
6:          u = extract_min(Q)
7:
8:          for each vertex v neighbour of u do
9:              R { if v ≠ s then
                    w = L_c from u to v
                    if MC[u][u] + w < MC[v][v] then
                      MC[v][v] = MC[u][u] + w
                    MC[u][v] = MC[u][u] + w
10:     return MC
```



Nodo	S	A	B	C	D	E	F
Padre	-	S	S	S	A	B	B
Coste	0	2	1	2	3	4	3

a) Algoritmo de Dijkstra.



Desde \ Hacia	S	A	B	C	D	E	F
S	0	2	1	2			
A		2	4		3		
B		3	1	2	8	4	3
C			3	2			4
D		4	10		3	5	
E			7		6	4	7
F			5	5		6	3

b) Análisis de costes de MDPAlg.

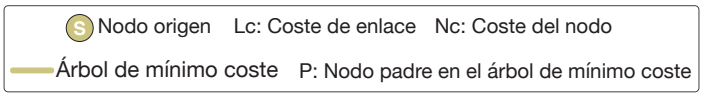


Figura 6.1: Analisis de costes.

mismos parámetros de entrada que el algoritmo de Dijkstra, el grafo topológico y el nodo origen, ubicando ambos algoritmos en el mismo punto de partida. Sin embargo, como se explicó anteriormente, MDPAlg unifica las estructuras de datos a utilizar, aglutinando toda la información relevante en la matriz *MC*. Por lo tanto, únicamente se inicializan la matriz *MC* y el vector *Q* (líneas 3 y 4), y se prescinde de los vectores *C* y *P*, ya que la información que contenían estas dos estructuras está incluida de forma indirecta en la matriz *MC*. Además, también se suprime el vector *S*, ya que ahora también se recalculan los costes de los nodos previamente visitados para caracterizar todos los *cross-links* al árbol de mínimo coste. El propósito del vector *Q* es el mismo que en el algoritmo de Dijkstra, asegurar que cada nodo solamente se visita una única vez, mientras que *MC* almacena el coste acumulador para ir desde el nodo origen a cualquier otro nodo considerando, tanto el árbol de mínimo coste, como los *cross-links*. La función *Initialize_single_source* realiza la misma misión que en el algoritmo de Dijkstra, establecer el coste del nodo origen a un valor 0 y asignar un valor infinito al resto de celdas de *MC*, para garantizar que los nodos son evaluados en un orden de coste ascendente. *MC* contiene en su diagonal el coste acumulado que presenta cada nodo a través del árbol de mínimo coste, equivalente al vector *C*, mientras que el resto de celdas almacenan el coste acumulado que obtienen las ramas alternativas al árbol de mínimo coste. De esta forma, una única estructura de datos almacena toda la información topológica obtenida durante el proceso de análisis de costes del grafo. Al igual que en el algoritmo de Dijkstra, la función de análisis de costes extrae el nodo de menor coste contenido en *Q* (u), y aplica la función RELAX a cada uno de sus vecinos (líneas 5 - 9). Esta función, siempre que el nodo vecino no sea el nodo origen, calcula el coste que tiene viajar desde el nodo origen hasta el nodo v atravesando el nodo u, e incluye dicho coste en la matriz *MC*. Además, si se reduce el coste de v, también almacena el coste obtenido en la diagonal de la matriz *MC*. Por lo tanto, el coste mínimo para llegar a cada nodo está duplicado, en la diagonal de la matriz y en la celda correspondiente al nodo vecino a través del que se llega. Esta duplicidad simplifica la operativa del algoritmo, ya que los valores almacenados en la diagonal son útiles para extraer del vector *Q* el nodo con menor coste en cada iteración sin tener que utilizar variables adicionales, mientras que la información de coste almacenada en la celda del nodo vecino se utiliza para la construcción de los caminos disjuntos. Finalmente, cuando se han visitado todos los nodos, el algoritmo devuelve la matriz *MC* con la caracterización completa de costes. El resultado de aplicar la función análisis de costes de MDPAlg sobre la topología de ejemplo queda representado en la Figura 6.1b, que muestra tanto el grafo topológico como la matriz *MC*. Dicha matriz, además de contener el coste acumulado para ir desde el nodo origen al resto de nodos de la topología a través de varias vías, incluye celdas marcadas en color gris que no son relevantes para la construcción de caminos disjuntos: la primera columna de la matriz y su diagonal principal. La primera columna almacena el coste acumulado que tiene viajar desde el nodo origen a si mismo, lo que no suscita interés porque no tiene sentido calcular caminos disjuntos con origen y destino

en el mismo nodo. Con respecto a la diagonal de la matriz, sus valores se encuentran duplicados, como se explicó anteriormente, por lo que no es necesaria dicha información en la fase de construcción de caminos. Un ejemplo del cálculo del coste acumulado se puede observar en la intersección de la fila $\boldsymbol{E}$ y la columna $\boldsymbol{F}$ de la matriz MC. El coste acumulado para viajar desde el nodo origen (S) hasta el nodo F es 7, cuyo resultado viene de la suma del coste del enlace entre los nodos E y F (Lc=*3*), más el coste acumulado para ir de S a E (4). Para el resto de celdas de la matriz se ha realizado el mismo proceso.

A la vista de los resultados, MDPAlg, con un procedimiento de cálculo similar al algoritmo de Dijkstra, es capaz de proporcionar un análisis de costes que caracteriza completamente al grafo topológico, incluyendo tanto el árbol de mínimo coste como información sobre los *cross-links*. Esta información adicional, será clave para encontrar los múltiples caminos disjuntos, ya que se tienen caracterizados todos los posibles caminos que comunican al nodo origen con el resto de nodos de la red.

6.1.2 Segunda fase: Construcción de caminos

Esta sección muestra el proceso seguido por MDPAlg para la construcción de múltiples caminos disjuntos entre un nodo origen y varios nodos destino, aprovechando la información recogida en la primera fase del algoritmo. MDPAlg, en esta segunda fase, interpreta la información contenida en la matriz MC para construir los múltiples caminos disjuntos, tarea que en el algoritmo de Dijkstra es mucho más sencilla, ya que construir el camino mínimo coste a partir de los vectores C y P es un proceso prácticamente directo.

La estructura que muestra el apartado es similar a la utilizada en la sección anterior, adjuntando el pseudocódigo de la fase de construcción de caminos (Algoritmo 6.3) y dos ilustraciones que describen visualmente el proceso de construcción de caminos para los modos de operación disjunto en enlaces y disjunto en nodos (Figuras 6.2 y 6.3). Ambas figuras muestran el proceso de construcción de caminos desde tres nodos frontera destino, los nodos E, D y F, siendo la Figura 6.2 la encargada de mostrar el proceso en el modo disjuntos en enlaces, y la Figura 6.3 de mostrarlo en modo disjuntos en nodos. Además, ambas figuras comparten estructura con la figura de la sección anterior, ubicando en la parte superior el proceso de construcción de caminos sobre el grafo topológico y en la parte inferior las operaciones realizadas en la MC para seleccionar los múltiples caminos disjuntos.

La función DisjointPathConstruction recogida en el Algoritmo 6.3, recibe como parámetros de entrada el grafo topológico (G), el nodo origen (s) y la Matriz de Costes (MC) devuelta por la primera fase. Este proceso de construcción de caminos disjuntos, al igual que en 1S-MDP, comienza en los nodos destino (obtenidos del grafo), y va construyendo cada camino disjunto utilizando la información contenida en MC hasta alcanzar el nodo origen. Como ya se ha explicado anteriormente, a pesar de que el proceso de construcción tiene una direccionalidad definida, los caminos obtenidos son

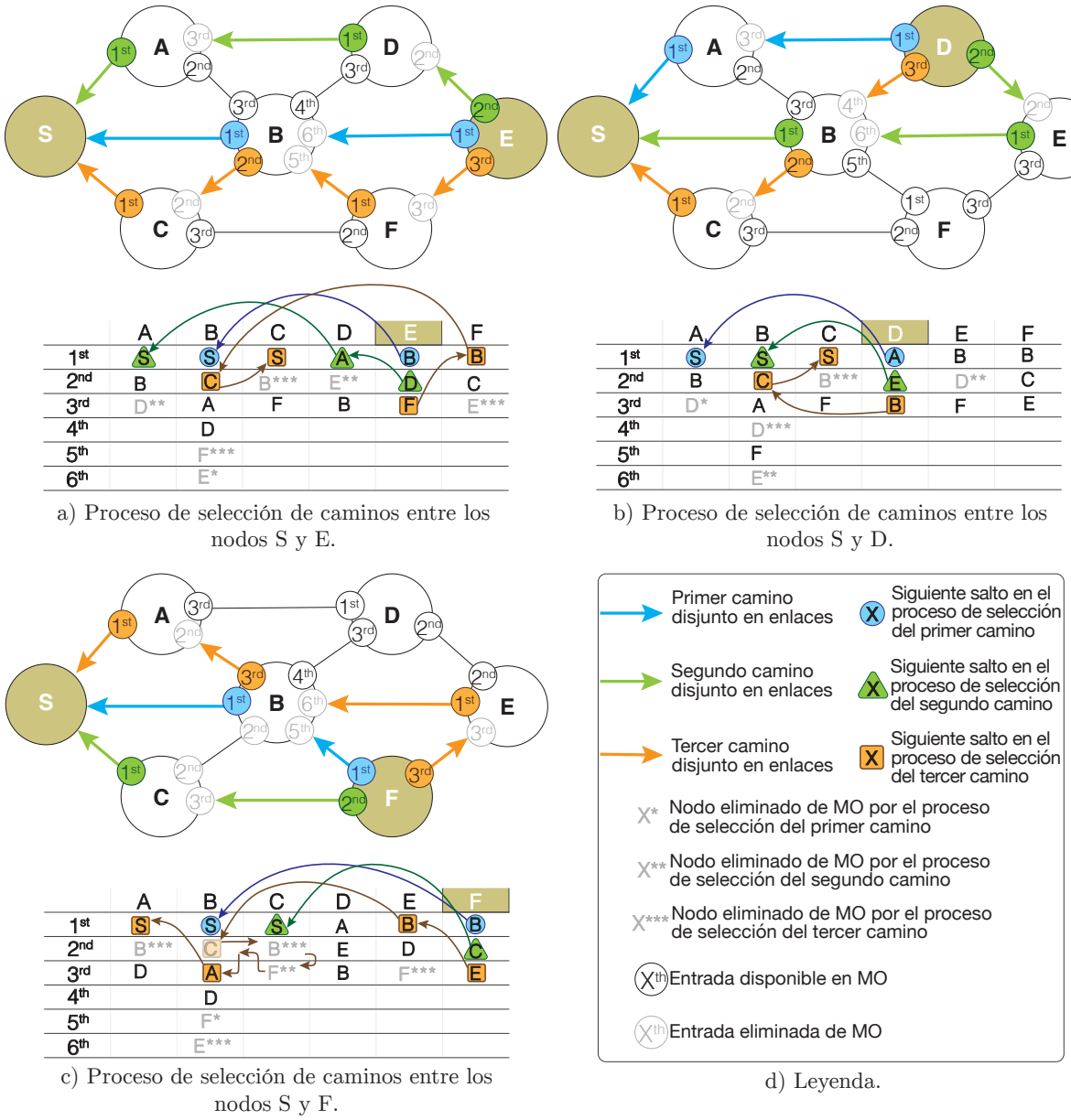
Algoritmo 6.3 Proceso de construcción de caminos disjuntos

```
1:  function DisjointPathConstruction(G, s, MC)
2:      Initialize_variables(P, MO)
3:      D = getDestinationNodes(G)
4:      while D ≠ ∅ do
5:          dst = extract node from D
6:          MO = MC
7:          while MO[dst] ≠ ∅ do
8:              new_path{}
9:              add dst to new_path
10:             Advance : previous = dst; next_hop = ExtractMinNeighbour(MO, previous)
11:             while next_hop ≠ s || next_hop ≠ ∅ do
12:                 add next_hop to new_path
13:                 if node − disjoint then
14:                     Remove next_hop from MO
15:                 else
16:                     Remove Link(next_hop, previous) from MO
17:                 Advance : previous = next_hop; next_hop = ExtractMinNeighbour(MO, previous)
18:             if next_hop == s then
19:                 add next_hop to new_path
20:                 if link − disjoint then
21:                     Remove possible loops in new_path
22:                 add new_path to P
23:             else
24:                 Hop Back : Remove previous from new_path; previous = getLastNode(new_path)
25:                 if previous ≠ dst then
26:                     Advance : next_hop = ExtractMinNeighbour(MO, previous)
27:                     Goto line 10
28:     return P
```

bidireccionales, ya que el grafo sobre el que se construyen es un grafo no dirigido. Además, MDPAlg, al tratarse de un algoritmo diseñado para trabajar en un entorno centralizado, en principio, no puede ejecutar el proceso de construcción de caminos desde varios nodos destino en paralelo, como hacía 1S-MDP, por lo que la función DisjointPathConstruction ejecutará de forma secuencial el proceso de construcción de caminos desde cada nodo destino. Esta función comienza inicializando la matriz MO y los vectores D y P (líneas 2 - 3). MO es una matriz con las mismas dimensiones de MC en la que se copia el contenido de MC al iniciar un nuevo proceso de construcción de caminos desde un nodo destino. Los vectores D y P, por su parte, se encargan de almacenar el conjunto de nodos destino desde los que hay que iniciar un proceso de construcción de caminos, y de guardar los caminos descubiertos, respectivamente. El conjunto de nodos destino se compone por un subconjunto de nodos del grafo o por la totalidad de sus nodos, excluyendo en cualquier caso al nodo origen. Tras obtener los nodos destino, la función extrae de D uno de sus nodos e inicia un proceso de construcción de caminos entre el nodo extraído y el nodo origen. Cuando este proceso termina, se repite la operativa hasta que el vector D queda vacío (líneas 4 - 5). En cada proceso de construcción de caminos entre el nodo origen y un nodo destino, se copia el contenido de la matriz MC en la matriz MO (línea 6). Esta copia se produce porque la información contenida MC es común para todos

procesos de construcción de caminos, sin embargo, cada proceso modifica unilateralmente la matriz sobre la que trabaja, por lo que la copia de los datos garantiza que todos los procesos de construcción de caminos reciben la información original. Desde cada nodo destino se buscan, de forma secuencial, rutas alternativas con el nodo origen a través sus vecinos. Para ello, el algoritmo almacena en la variable *previous* el nodo destino (*dst*) y extrae de *MO* el vecino del nodo destino con menor coste ejecutando la función *ExtractMinNeighbour* (líneas 7 - 9). Esta función recibe una matriz de coste y un nodo, y extrae de dicha matriz el nodo vecino del nodo proporcionado que presenta menor coste. De esta forma se avanza hacia el nodo origen guardando una trazabilidad del camino que se está construyendo, pudiendo retroceder hacia pasos previos en caso de que no se encuentre una ruta válida. El proceso continúa extrayendo nodos de *MO* con ayuda de la función *ExtractMinNeighbour* hasta que se alcanza el nodo origen o no se encuentra un nodo válido, actualizando además las variables *previous* y *next_hop*, y añadiendo los nuevos nodos al camino (líneas 10 -17). Además, en función del tipo de caminos que se están construyendo (disjuntos en enlaces o en nodos), se realizan una serie de pasos adicionales (líneas 13 - 16). Si el modo de operación es disjunto en nodos, al elegir un nuevo nodo *next_hop*, éste se elimina por completo de la matriz *MO*, así ningún otro camino entre el mismo par de nodos origen-destino puede elegirlo en el proceso de construcción de caminos. Si por el contrario, el modo de operación es disjunto en enlaces, solamente se deben eliminar de la matriz *MO* las celdas que describen el enlace entre los nodos *next_hop* y *previous* en ambas direcciones, puesto que los caminos son bidireccionales y dos caminos disjuntos en enlaces entre el mismo par de nodos no pueden compartir enlaces entre sí. Cuando el proceso de construcción llega al nodo origen, se añade dicho nodo al camino *new_path*, y el camino *new_path* se añade al vector que almacena el conjunto de caminos disjuntos obtenidos (P) (líneas 18 - 22). Sin embargo, si los caminos obtenidos son disjuntos en enlaces, se debe comprobar que el camino no tenga lazos internos, ya que se puede dar la casuística que, durante el proceso de construcción del camino, un nodo se visite dos veces. En este caso tan peculiar, se corta el lazo interno y se deja al camino libre de lazos. También se puede dar el caso de que la función *ExtractMinNeighbour* no encuentre un nodo vecino válido. En este caso, la función retrocede un paso atrás (al nodo previo en el camino que se está construyendo) para buscar una vía alternativa que le permita llegar hasta el nodo origen (líneas 23 - 27). Se puede retroceder tantos pasos como se necesiten, siempre que en el proceso de retroceso no se alcance el nodo destino.

Para comprender mejor todo el proceso, se adjuntan las Figuras 6.2 y 6.3, quienes muestran el proceso de construcción de caminos para el modo disjunto en enlaces y disjunto en nodos, respectivamente. Ambas figuras contienen en la parte superior una representación de los pasos que sigue el proceso de construcción de caminos sobre el grafo de la topología, mientras que la parte inferior ilustra la secuencia de operaciones que realiza el algoritmo sobre la matriz *MO* para construir los múltiples caminos disjuntos. Como se ha explicado anteriormente, la función DisjointPathConstruction copia el



a) Proceso de selección de caminos entre los nodos S y E.

b) Proceso de selección de caminos entre los nodos S y D.

c) Proceso de selección de caminos entre los nodos S y F.

d) Leyenda.

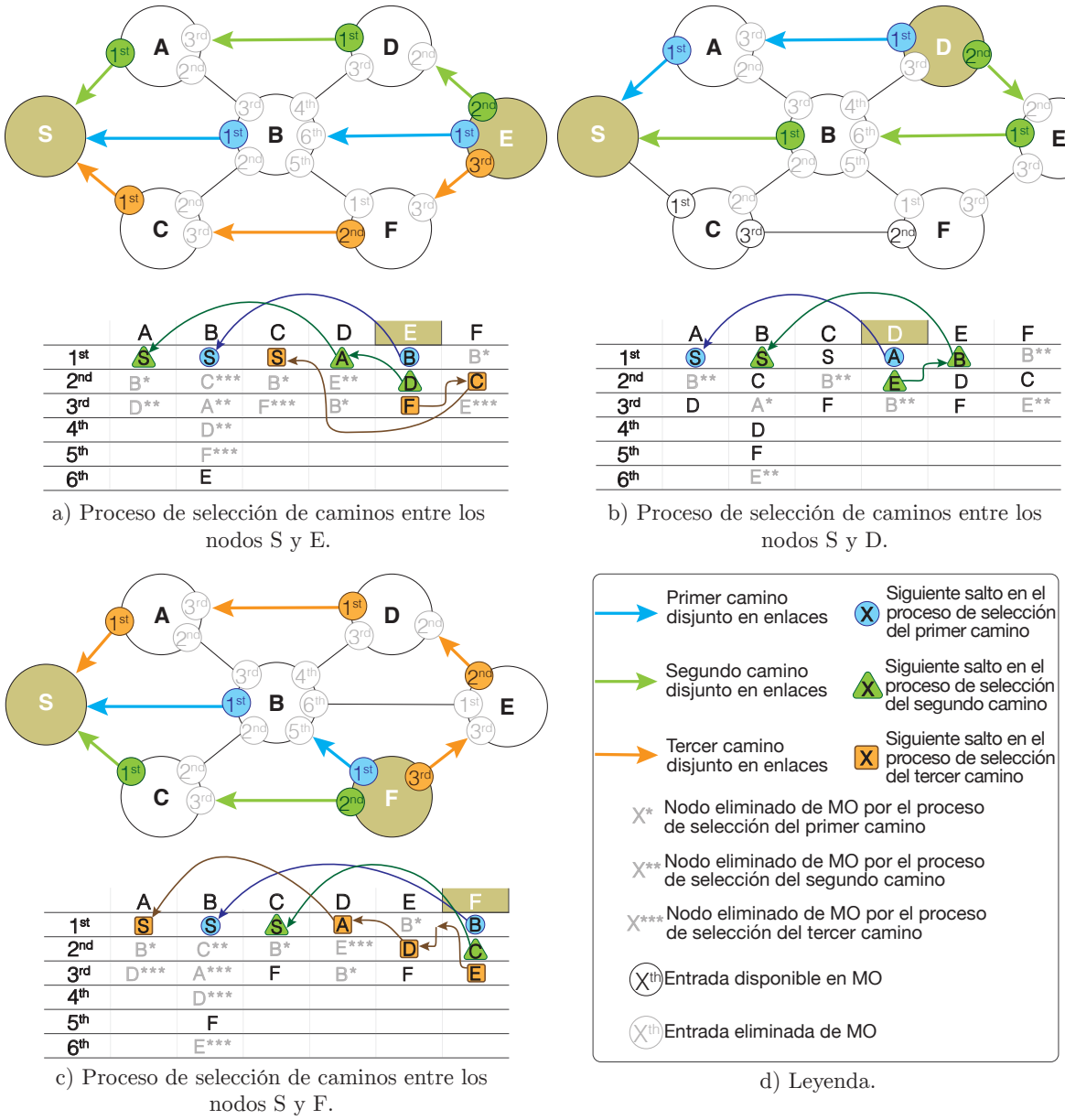
Figura 6.2: Construcción de caminos en modo disjuntos en enlaces.

contenido de la matriz *MC* en la matriz *MO*, sin embargo, la matriz *MO* representada en las figuras no comparte estructura con la matriz *MC*. Esto se debe a que en la figura se ha ilustrado la matriz *MO* con la ordenación que devuelve la función *ExtractMinNeighbour*, que devuelve los nodos en función del coste que tienen acumulado desde el nodo origen. En cada columna de la matriz se muestran los nodos vecinos de cada nodo, ordenados en coste ascendente, ocupando así las primeras posiciones los nodos que presentan menor coste. Cuanto más abajo esté situado el nodo vecino, peor será el coste que obtuvo en la fase de análisis de costes. Además, la función *ExtractMinNeighbour* incluye el sistema de modificación de prioridades descrito en la Sección 5.3.3 del capítulo anterior, para evitar problemas de lazos cerrados que impedían completar el proceso de construcción de caminos con normalidad. Esta modificación se aplicaba a las interfaces de los switches que iban en sentido contrario al árbol de mínima latencia obtenido durante la fase de

exploración de 1S-MDP. Aquí, siguiendo la misma analogía, deben ser penalizados los enlaces que van en sentido contrario al árbol de mínimo coste. Por ejemplo, el proceso de análisis de coste realizado obtiene que el árbol de mínima latencia pasa por B y se dirige a E y a F, por lo tanto, las entradas de la matriz MO que describen los enlaces en sentido inverso deben ser penalizadas. El proceso de ordenación por coste original debería haber ordenado la columna B de la matriz MO en el siguiente orden [S C A F E D], sin embargo, el orden final que muestra MO es [S C A $\boldsymbol{D}$ $\boldsymbol{F}$ $\boldsymbol{E}$] debido a que el coste de los enlaces para ir desde E y F hacia B han sido modificados, por lo que se ubican en las últimas posiciones de la columna. El mismo fenómeno ocurre en la columna A, donde el coste del nodo D se modifica por ir en sentido contrario al árbol de mínimo coste.

La Figura 6.2 muestra el proceso de construcción de caminos en modo disjunto en enlaces entre tres pares de nodos distintos: S y E (Figura 6.2a), S y D (Figura 6.2b), y S y F (Figura 6.2c). En todas ellas el proceso de construcción de caminos comienza desde el nodo destino (E, D y F, respectivamente) y acaba en el nodo origen S. Concretamente, en la Figura 6.2a el nodo E comienza el proceso de construcción de caminos seleccionando el nodo vecino con menor coste que tiene disponible, el nodo B, que, a su vez, elige su nodo vecino que presenta menor coste, el nodo S, finalizando así el proceso de construcción del primer camino disjunto entre los nodos S y E. Durante este proceso se han ido eliminando de la matriz MO las celdas que identifican los enlaces utilizados por el primer camino disjunto en ambos sentidos (de origen a destino y de destino a origen), como se explicó anteriormente. Estas celdas están representadas en color gris claro en la matriz MO y el asterisco que las acompaña indica el camino que las eliminó de la matriz. De esta forma se garantiza la disjuntividad en enlaces con caminos posteriores, ya que éstos no pueden volver a seleccionar los enlaces utilizados por el primer camino. Cuando finaliza el proceso de construcción del primer camino, comienza el del segundo camino. El nodo E, selecciona el vecino de menor coste disponible, D, puesto que el enlace de E a B se utilizó en el primer camino. El nodo D repite la operativa, y selecciona como vecino de menor coste al nodo A quien, a su vez, elige a S, por lo que finaliza el proceso de construcción del segundo camino. En el tercer camino se vuelve a repetir el procedimiento, seleccionando en cada paso el vecino que presenta un coste menor. No obstante, cuando finaliza el proceso de construcción de este tercer camino, el algoritmo inicia el proceso de construcción de caminos desde el siguiente nodo destino (Figura 6.2b), ya que no existen más caminos disjuntos posibles entre los nodos S y E. La metodología que sigue el proceso para construir los caminos disjuntos entre los nodos S y D es idéntica al proceso de construcción de caminos disjuntos entre los nodos S y E, seleccionando en cada salto el nodo vecino con menor coste. Por último, la Figura 6.2c recoge el proceso de construcción de caminos entre los nodos S y F, cuyo procedimiento es análogo al de las dos figuras anteriores.

La metodología cambia ligeramente cuando los caminos construidos son disjuntos en nodos, como recoge la Figura 6.3. Si bien el sistema para elegir el siguiente salto no



a) Proceso de selección de caminos entre los nodos S y E.

b) Proceso de selección de caminos entre los nodos S y D.

c) Proceso de selección de caminos entre los nodos S y F.

d) Leyenda.

Figura 6.3: Construcción de caminos en modo disjuntos en nodos.

varía (el vecino con menor coste), el borrado de elementos en la matriz *MO* sí que cambia. Ahora, además de borrar el camino construido, se debe eliminar de la matriz *MO* cualquier celda que permita llegar de forma directa a un nodo que pertenece a un camino disjunto, exceptuando los nodos extremos. De esta forma se evita que caminos disjuntos posteriores elijan un nodo que pertenece a un camino disjunto previamente construido, garantizando así que cada nodo solamente contiene un camino disjunto entre el mismo par de nodos origen-destino. Para entender mejor este concepto, se pone de ejemplo la Figura 6.3a, donde el proceso de construcción del primer camino elige al nodo *B* como nexo de unión entre los nodos *S* y *E*. Cuando se elige a *B*, al mismo tiempo, se elimina de la matriz *MO* cualquier entrada que contenga una referencia al nodo *B*, de esta forma, caminos posteriores no pueden elegir el nodo *B* como siguiente salto. Este

borrado selectivo hace que, por ejemplo, el proceso de construcción del tercer camino, cuando llega al nodo F no puede elegir al nodo B, aunque fuese el vecino de menor coste, por lo que el nodo F tiene que elegir al nodo C como siguiente salto, cuyo resultado final son tres caminos disjuntos en nodos entre S y E. Por último, las Figuras 6.3b y 6.3c muestran sendos procesos de construcción de caminos para los otros dos nodos destino, donde se observa que, debido a la restricción impuesta para garantizar caminos disjuntos en nodos, el proceso de construcción de caminos entre los nodos S y D obtiene solamente dos caminos disjuntos. De hecho, cualquier proceso de búsqueda de caminos disjuntos en nodos, a menudo, encontrará un menor número de caminos disjuntos en nodos que enlaces debido, precisamente, a las restricciones que presenta el modo disjunto en nodos. Sin embargo, a pesar de poder encontrar menos caminos, éstos son de mayor calidad, ya que no comparten entre sí ningún elemento a excepción de los nodos extremos.

En cuanto al sistema de salto atrás ejecutado cuando la función *ExtractMinNeighbour* no encuentra un vecino válido, éste queda representado en la Figura 6.2c, durante el proceso de construcción del tercer camino. Cuando el proceso de construcción llega al nodo B, la función *ExtractMinNeighbour* devuelve como vecino de menor coste el nodo C sin embargo, el nodo C no tiene vecinos disponibles, ya que éstos se eliminaron de la matriz MO durante el proceso de construcción del primer y del segundo camino disjuntos. Por lo tanto, al ejecutar de nuevo la función *ExtractMinNeighbour* en el nodo C, el resultado que se obtiene es nulo. Al no encontrar un vecino válido, el algoritmo retrocede al nodo anterior (B) y vuelve a ejecutar la función *ExtractMinNeighbour* para encontrar un vecino alternativo, devolviendo como resultado el siguiente nodo en la lista, el nodo A. A través del nodo A el proceso de construcción alcanza el nodo origen, por lo que el proceso de construcción de caminos termina satifactoriamente. Sin embargo, dicha operación puede verse bloqueada, como ocurre en la Figura 6.3b. En este escenario, la construcción del tercer camino directamente no puede ejecutarse, ya que el nodo B se utilizó en el segundo camino y no puede ser utilizado por ningún otro camino, lo que condena al proceso de construcción del tercer camino desde el nodo D.

6.2 Complejidad computacional

La complejidad computacional es un término utilizado en ciencias de computación para determinar la eficiencia de los algoritmos en función de los recursos temporales y físicos que consumen en su ejecución, lo que constituye un buen estimador para cuantificar la calidad de las propuestas. Cuanto menor sea su valor más eficiente será la propuesta, ya que el problema formulado se puede resolver con una cantidad finita de recursos. En caso contrario, el consumo de recursos se incrementa, pudiendo llegar al extremo de que el problema no puede ser resuelto por un sistema de cómputo.

Por lo tanto, estudiar la complejidad computacional de MDPAlg bajo un escenario

extremo constituye una medida objetiva de la calidad de la propuesta, ya que se puede estimar la cantidad de recursos que consume el algoritmo para resolver dicho problema. Sin embargo, este algoritmo se debe comparar con otros trabajos que sirvan como referencia, para ver si los resultados obtenidos mejoran a los previamente publicados. Aquí es donde entra el juego el algoritmo de Dijkstra, ya que es un referente utilizado a lo largo de los años con el que se pueden calcular caminos disjuntos sobre un grafo de la red aplicando técnicas de recursividad. Además, este algoritmo también se utiliza como algoritmo base sobre el que MDPAlg construye su proceso de análisis de costes, por lo que es el algoritmo ideal para desarrollar el estudio teórico de la complejidad computacional. No obstante, al final de la sección se extenderá el estudio a otras propuestas, pero mostrando solamente el resultado final para no alargar en exceso la longitud de este capítulo.

Dado que esta sección requiere formulación matemática para realizar el análisis teórico, en primer lugar, se introduce dicha formulación y se describe matemáticamente el problema a resolver. Una vez descrito el problema, se analiza con el algoritmo de Dijkstra, y, posteriormente, con el algoritmo MDPAlg. Al final se muestran los resultados de ambas propuestas y se expande el estudio a otros trabajos.

6.2.1 Formulación matemática del problema

Antes de embarcarse en el estudio teórico de la complejidad computacional, se comprobó que dicho estudio fuese viable. El problema que trata la búsqueda de caminos disjuntos encaja en la clase NP-completo [Middendorf & Pfeiffer, 1993, Robertson & Seymour, 1995, Eilam-Tzoreff, 1998], lo que garantiza que se puede resolver en tiempo polinómico mediante cálculos basados en fuerza bruta, por lo que existe una solución al problema cuyo tiempo de resolución dependerá de una función polinómica.

La formulación matemática del problema de búsqueda de múltiples caminos disjuntos sobre un grafo topológico se define de la siguiente forma:

Dado el grafo $G = (\mathcal{N}, \mathcal{L})$, formado por un conjunto de $\mathcal{N}$ de N nodos, un conjunto $\mathcal{L}$ de L enlaces, y dos nodos $s, t \in \mathcal{N}$. Se buscan k caminos disjuntos $P_1, P_2, ..., P_k$ entre los nodos s y t siendo s el nodo origen y t el nodo destino. Además, si los caminos son disjuntos en enlaces $P_1, P_2, ..., P_k$ no comparten entre sí ningún enlace $l \in \mathcal{L}$. Mientras que si los caminos son disjuntos entre nodos, $P_1, P_2, ..., P_k$ no comparten entre sí ningún enlace $l \in \mathcal{L}$ ni ningún nodo $n \in \mathcal{N}$, a excepción de los nodos origen y destino s y t.

Además, se quiere llevar al caso extremo el estudio teórico, por lo que se utiliza la topología que contiene el mayor número de enlaces por cada nodo, un grafo en el que todos los nodos están conectados entre sí. De esta forma, los algoritmos tendrán que analizar el número máximo de enlaces posibles que puede tener una red, llevando al límite el proceso de cómputo. Análogamente, la búsqueda de caminos disjuntos no se limita solamente a la búsqueda de caminos entre un par de nodos, sino que se buscan todos los caminos disjuntos posibles entre todos los pares de nodos de la topología para incrementar

la cantidad de operaciones que tiene que ejecutar cada algoritmo. Bajo estas premisas, el problema de la búsqueda de caminos disjuntos queda definido de la siguiente forma:

Dado el grafo más extremo, $G_E = (\mathcal{N}, \mathcal{L})$, formado por un conjunto de $\mathcal{N}$ de N nodos, un conjunto $\mathcal{L}$ de L enlaces, en el cual, cada nodo $n_i \in \mathcal{N}$ está conectado con todos los nodos $n_j \in \mathcal{N}\ \forall\ n_i \neq n_j$; el conjunto de nodos origen S = $\mathcal{N}$ y el conjunto de nodos destino T = $\mathcal{N}$, se buscan todos los posibles caminos disjuntos entre los pares de nodos $\{s_j, t_i\}$ / $s_j \neq t_i$, con $0 < i \leqslant N$ y $0 < j \leqslant N$, cuyo resultado es $P_{1(s_j,t_i)}, P_{2(s_j,t_i)}, ...P_{k(s_j,t_i)}, 0 < k \leqslant N-1$.

6.2.2 Complejidad computacional del algoritmo de Dijkstra

Como se ha visto a lo largo de la Tesis, el algoritmo de Dijkstra devuelve dos vectores, un vector de nodos padre y un vector de costes, a través de los que se puede obtener el árbol de mínimo coste. Dicho árbol proporciona múltiples rutas, una ruta entre el nodo origen y cada uno de los nodos restantes que forman la red. Sin embargo, si se quieren obtener múltiples caminos disjuntos entre un nodo origen y un nodo destino, el algoritmo de Dijkstra se tiene que ejecutar de forma recursiva, eliminando en cada iteracción el camino de menor coste obtenido entre los nodos origen y destino. Así se garantiza que cada búsqueda obtiene un camino disjunto al previamente calculado entre el mismo par de nodos. Si se eliminase del grafo el árbol de mínimo coste al completo, cada ejecución obtendría árboles disjuntos entre sí, pero no se descubriría la totalidad de caminos disjuntos entre un par de nodos, ya que se reduce el número de enlaces disponibles para el cálculo de caminos entre dicho par de nodos, al ser utilizados por las ramas de los árboles. Por este motivo, el algoritmo de Dijkstra se tiene que ejecutar tantas veces como caminos disjuntos se quieran descubrir entre un par de nodos origen-destino, eliminando, en cada iteración, el camino de mínimo coste descubierto entre dicho par de nodos. Además, este proceso se tiene que repetir tantas veces como pares de nodos origen-destino distintos existan en la topología, puesto que se quieren descubrir todos los caminos disjuntos existentes entre todos los pares de nodos.

Teorema 6.2.1. *La complejidad computacional del algoritmo de Dijkstra para obtener el árbol de mínimo coste entre un par de nodos es proporcional al número de nodos del grafo al cuadrado, $O(N^2)$ [Cormen et al., 2001]. Por lo tanto, en el grafo extremo G_E, en el que todos los nodos están conectados entre sí, la complejidad computacional que presenta el algoritmo de Dijkstra para descubrir todos los caminos disjuntos entre un par de nodos $\{s, t\}$, $s \in S$, $t \in T$, $s \neq t$ es $O(N^3)$.*

Demostración. El algoritmo de Dijkstra se tiene que ejecutar tantas veces como caminos disjuntos se quieren obtener entre un par de nodos. En el grafo extremo G_E todos los nodos están conectados entre sí, por lo tanto, teniendo en cuenta que dicho grafo está compuesto por N nodos, cada nodo $n_i \in \mathcal{N},\ 0 < i < N$ está conectado con $N-1$ vecinos. Así pues,

existen $N-1$ posibles caminos disjuntos entre cada par de nodos. En consecuencia, para obtener todos los caminos disjuntos entre un par de nodos $\{s,t\},\ s \in S,\ t \in T,\ s \neq t$ $(P_{1(s,t)}, P_{2(s,t)}, ...P_{k(s,t)}; 0 < k \leqslant N-1)$, el algoritmo de Dijkstra se tiene que ejecutar tantas veces como posibles caminos disjuntos tiene el par de nodos $\{s,t\}$, $N-1$ veces. Dado que una ejecución del algoritmo de Dijkstra tiene una complejidad computacional $O(N^2)$, al ejecutarlo $N-1$ veces la complejidad computacional resultante que se obtiene es $O(N^2) \cdot (N-1) \simeq O(N^3)$ □

Teorema 6.2.2. *La complejidad computacional del algoritmo de Dijkstra para descubrir todos los caminos disjuntos entre un nodo origen $s \in S$ y todos los nodos destinos $t_i \in T,\ \forall\, t_i \neq s,\ 0 < i \leqslant N$ en el grafo extremo G_E es $O(N^4)$.*

Demostración. El Teorema 6.2.1 demuestra que la complejidad computacional del algoritmo de Dijkstra para obtener todos los caminos disjuntos entre un nodo origen $s \in S$, y un nodo destino $t \in T,\ s \neq t$ en un grafo extremo G_E, es $O(N^3)$. Por lo tanto, para descubrir todos los caminos disjuntos entre un nodo origen $s \in S$, y todos los nodos destinos posibles $t_i \in T,\ \forall\, t_i \neq s,\ 0 < i \leqslant N$ $(P_{1(s,ti)}, P_{2(s,ti)}, ...P_{k(s,ti)}; 0 < k \leqslant N-1))$, el proceso descrito en el Teorema 6.2.1 se tiene que repetir tantas veces como nodos destino t_i tenga el nodo origen s. Como cada un nodo origen tiene $N-1$ nodos vecino destino, el proceso se tiene que repetir $N-1$ veces, dando una complejidad computacional resultante $O(N^3) \cdot (N-1) \simeq O(N^4)$ □

Teorema 6.2.3. *En un grafo extremo G_E la complejidad computacional del algoritmo de Dijkstra para obtener todos los caminos disjuntos entre todos los pares de nodos es $O(N^5)$.*

Demostración. La complejidad computacional que obtiene el algoritmo de Dijkstra en el grafo G_E para obtener todos los caminos disjuntos entre un nodo origen $s \in S$ y todos los nodos destinos $t_i \in T,\ \forall\, t_i \neq s,\ 0 < i \leqslant N$ es $O(N^4)$, según recoge el Teorema 6.2.1. Por lo tanto, para obtener los caminos disjuntos entre todos los pares de nodos posibles, se tienen que repetir los pasos del Teorema 6.2.1 tantas veces como nodos origen existan en G_E. El conjunto de nodos origen de G_E es S, compuesto por N nodos, por lo tanto, se tiene que volver a ejecutar N veces el algoritmo de Dijkstra, resultando una complejidad computacional $O(N^4) \cdot N = O(N^5)$. □

Del estudio teórico realizado se recoge que la complejidad computacional del algoritmo de Dijkstra para obtener todos los caminos disjuntos entre todos los pares de nodos de la red en una topología extrema, en la que todos los nodos están conectados entre sí, es $O(N^5)$. Este resultado indica que el tiempo que tarda el algoritmo de Dijkstra en resolver el problema planteado depende del número de nodos (N) que tenga la topología, aumentando de forma exponencial el tiempo invertido por Dijkstra para resolver el problema conforme se incrementa el número de nodos de dicha topología. La función resultante será del tipo $tiempo\_resolución = a \cdot n^5$, dónde a es una constante y n el número de nodos de la topología.

6.2.3 Complejidad computacional de MDPAlg

El estudio de la complejidad computacional de MDPAlg sigue la misma metodología que el estudio teórico realizado para el algoritmo de Dijkstra, dónde se analiza el caso más sencillo y se va incrementando la complejidad del problema hasta llegar a la solución buscada. En este análisis también se trabaja sobre G_E para obtener todos los caminos disjuntos posibles entre todos los pares de nodos.

El algoritmo MDPAlg se compone de un proceso dividido en dos fases, una primera fase de análisis de costes seguida por una segunda fase de construcción de caminos, pudiéndose calcular con estos dos pasos, todos los caminos disjuntos entre un nodo origen y el resto de nodos del grafo. Puesto que ambas fases se corresponden con procesos independientes que se ejecutan en cascada, la complejidad computacional de cada fase se va a analizar por separado. El resultado final de la complejidad computacional de MDPAlg viene determinado por la suma de la complejidad computacional de ambas fases. Además, debido a las diferencias que existen en la fase de construcción de los caminos disjuntos en función del modo de trabajo (caminos disjuntos en enlaces o disjuntos en nodos), cada uno de los modos también se analiza por separado.

Teorema 6.2.4. *La fase de análisis de costes de MDPAlg presenta una complejidad computacional cuadrática, $O(N^2)$, para realizar un análisis de costes del grafo completo, y una complejidad computacional cúbica, $O(N^3)$, para ejecutar todos los procesos de análisis de costes necesarios para la obtención de todos los caminos disjuntos entre todos los pares de nodos en el grafo G_E.*

Demostración. La fase de análisis de costes de MDPAlg utiliza la misma estructura de búsqueda que el algoritmo de Dijkstra para realizar el cálculo de costes, solo que en lugar de almacenar únicamente información del camino de mínimo coste, también guarda información del resto de alternativas. Por lo tanto, MDPAlg no añade nuevas operaciones que impliquen la evaluación de nuevos nodos o enlaces, por lo que la complejidad computacional resultante de realizar el análisis de costes es igual a la complejidad computacional que supone ejecutar el algoritmo de Dijkstra $O(N^2)$ [Cormen et al., 2001]. Además, con un único análisis de coste se obtienen caminos entre un nodo origen $s \in S$ y todos los nodos destinos posibles $t_i \in T$, $\forall\, t_i \neq s$, $0 < i \leqslant N$ $(P_{1(s,ti)}, P_{2(s,ti)}, ...P_{k(s,ti)}; 0 < k \leqslant N-1))$, por lo que el proceso de análisis de costes, para obtener todos los caminos disjuntos entre todos los pares de nodos del grafo, se tiene que repetir tantas veces como nodos origen tiene G_E. En el ejemplo bajo estudio el conjunto de nodos origen, S, contiene todos los nodos del grafo G_E, por lo que la fase de análisis de costes tiene que ejecutarse N veces. Por lo tanto, la complejidad computacional resultante para ejecutar los N procesos de análisis de costes que permiten obtener todos los caminos múltiples disjuntos entre $\{s_j, t_i\}$, $s_j \in S$, $t_i \in T$, $\forall\, s_j \neq t_i$; $0 < \{j, i\} \leqslant N$ $(P_{1(sj,ti)}, P_{2(sj,ti)}, ...P_{k(sj,ti)}; 0 < k \leqslant N-1)$, es $O(N^2) \cdot N = O(N^3)$. □

Teorema 6.2.5. *La fase de construcción de caminos, en modo caminos disjuntos en enlaces, sobre un grafo extremo G_E presenta una complejidad computacional $O(N^4)$ para descubrir todos los caminos disjuntos entre todos los pares de nodos $\{s_j, t_i\}$, $s_j \in S$, $t_i \in T$, $\forall\ s_j \neq t_i$; $0 < \{j, i\} \leqslant N$.*

Demostración. En el proceso de construcción de los caminos disjuntos, la peor situación, en términos de complejidad computacional, ocurre cuando, entre un par de nodos $s, t \in \mathcal{N}$, el conjunto de caminos disjuntos obtenidos utiliza todos los enlaces de la topología, puesto que el algoritmo tiene que procesar cada uno de esos enlaces. El conjunto de enlaces en un grafo extremo WC_G, viene determinado por la suma de los enlaces que tiene cada nodo, por lo que, si el grafo tiene N nodos, y cada nodo tiene N-1 vecinos, al estar todos conectados entre sí, el número total de enlaces será $N \cdot (N-1) = N^2 - N$. Sin embargo, como estos enlaces son bidireccionales, en realidad el grafo WC_G tiene la mitad de enlaces, ya que un mismo enlace contiene ambos sentidos de la comunicación, por lo que el número de enlaces de WC_G es $\frac{N^2-N}{2}$. En consecuencia, la complejidad computacional de la peor situación (cuando los caminos disjuntos entre un par de nodos utilizan todos los enlaces de la topología) es proporcional al número de enlaces que se tienen que procesar, lo que da como resultado una complejidad computacional cuadrática, $O(\frac{N^2-N}{2}) \simeq= O(N^2)$. Si se produce el mismo fenómeno por cada conjunto de caminos disjuntos obtenidos entre un nodo origen $s \in S$ y el resto de nodos destinos de G_E, $t_i \in T$, $\forall t_i \neq s$, $0 < i \leqslant N$, la situación anterior se repite tantas veces como nodos destino tiene cada nodo origen, $N-1$ veces. Por lo tanto, la complejidad computacional resultante de obtener múltiples caminos disjuntos entre un nodo origen $s \in S$ y todos los nodos destinos de G_E, $t_i \in T$, $\forall t_i \neq s$, $0 < i \leqslant N$ $(P_{1(s,ti)}, P_{2(s,ti)}, ...P_{k(s,ti)}; 0 < k \leqslant N-1))$, es $O(N^2) \cdot N = O(N^3)$. Si la situación se reproduce en todos los pares de nodos entre los que se calculan los caminos disjuntos $\{s_j, t_i\}$ / $s_j \neq t_i$, con $0 < i \leqslant N$ y $0 < j \leqslant N$ $(P_{1(s_j,t_i)}, P_{2(s_j,t_i)}, ...P_{k(s_j,t_i)}, 0 < k \leqslant N-1)$, el paso anterior se ha de repetir tantas veces como nodos origen haya en G_E (N). Por lo tanto, la complejidad computacional de la fase de construcción de caminos en el modo caminos disjuntos en nodos es $O(N^3) \cdot N = O(N^4)$. □

Teorema 6.2.6. *La fase de construcción de caminos, en modo caminos disjuntos en nodos, sobre un grafo extremo G_E presenta una complejidad computacional $O(N^3)$ para descubrir todos los caminos disjuntos entre todos los pares de nodos $\{s_j, t_i\}$, $s_j \in S$, $t_i \in T$, $\forall\ s_j \neq t_i$; $0 < \{j, i\} \leqslant N$.*

Demostración. En el proceso de construcción de los caminos disjuntos, la peor situación, en términos de complejidad computacional, ocurre cuando, entre un par de nodos $s, t \in N$, se utilizan todos los nodos disponibles en la topología por el conjunto de caminos disjuntos, puesto que el algoritmo tiene que procesar cada uno de esos nodos. En un grafo extremo WC_G, el conjunto de nodos es N, por lo tanto, obtener todos los caminos disjuntos entre un par de nodos s, t es proporcional a N, $O(N)$. Si se produce el mismo

fenómeno por cada conjunto de caminos disjuntos obtenidos entre un nodo origen $s \in S$ y el resto de nodos destinos de G_E, $t_i \in T$, $\forall t_i \neq s$, $0 < i \leqslant N$, la situación anterior se repite tantas veces como nodos destino tiene cada nodo origen, $N-1$ veces. Por lo tanto, la complejidad temporal resultante de obtener múltiples caminos disjuntos entre un nodo origen $s \in S$ y todos los nodos destinos de G_E, $t_i \in T$, $\forall t_i \neq s$, $0 < i \leqslant N$ $(P_{1(s,ti)}, P_{2(s,ti)}, ...P_{k(s,ti)}; 0 < k \leqslant N-1))$, es $O(N) \cdot N = O(N^2)$. Si la situación se reproduce en todos los pares de nodos entre los que se calculan los caminos disjuntos $\{s_j, t_i\}$ / $s_j \neq t_i$, con $0 < i \leqslant N$ y $0 < j \leqslant N$ $(P_{1(s_j,t_i)}, P_{2(s_j,t_i)}, ...P_{k(s_j,t_i)}, 0 < k \leqslant N-1)$, el paso anterior ha de repetir tantas veces como nodos origen haya en G_E (N). Por lo tanto, la complejidad computacional de la fase de construcción de caminos en el modo caminos disjuntos en nodos es $O(N^2) \cdot N = O(N^3)$. □

Finalmente, la complejidad computacional de MDPAlg viene determinada por la suma de la complejidad computacional de sus dos fases (análisis de costes y construcción del camino). Para el modo disjunto en enlaces el resultado es $O(N^3) + O(N^4) \simeq O(N^4)$, mientras que para el modo disjunto en nodos es $O(N^3) + O(N^3) \simeq O(N^3)$. Por lo tanto, MDPAlg reduce en uno o dos órdenes de magnitud la complejidad computacional con respecto al algoritmo de Dijkstra en función del modo de operación.

6.2.4 Extensión del análisis

El objetivo de esta sección es extender el estudio teórico de la complejidad computacional a otras propuestas, para verificar que MDPAlg es un algoritmo eficiente que reduce la complejidad computacional con respecto a otras soluciones encontradas en la literatura. Sin embargo, dada la extensión de cada uno de los análisis, este apartado simplemente recoge los resultados de todas las propuestas y las representa en forma de tabla, para comparar rápidamente su complejidad computacional. Todos los análisis teóricos se han llevado a cabo bajo las mismas condiciones, calculando la complejidad computacional para obtener todos los caminos disjuntos posibles entre todos los pares de nodos disponibles en el grafo G_E. En primer lugar, se extendió el análisis a implementaciones conocidas del algoritmo de Dijkstra que optimizan su proceso de cálculo, como son las implementaciones con cola de prioridad o con montículo de Fibbonaci y, después, se prolongó el estudio a otras propuestas centradas exclusivamente en el cálculo de caminos disjuntos.

La Tabla 6.1 recoge el resultado del estudio teórico, mostrando en cada una de las filas la complejidad computacional de cada algoritmo, clasificada por columnas en función del número de caminos obtenidos. Además, se indica el tipo de caminos disjuntos descubiertos, disjunto en nodos (DN), disjunto en enlaces (DE) y sin disjuntividad (SD). Por ejemplo, la primera fila recoge los resultados de MDPAlg en modo nodo (DN), y la primera columna recopila la complejidad computacional resultante de obtener un camino disjunto entre un par de nodos. Si alguna celda de la matriz está vacía, significa que dicho algoritmo como mínimo obtiene más caminos de los que indica el encabezado de la tabla. Volviendo al

Tabla 6.1: Complejidad computacional.

	Complejidad computacional			
	Un camino entre $s, t \in \mathcal{N}$ $\forall\ s \neq t$	Caminos entre $s, t \in \mathcal{N}$ $\forall\ s \neq t$	Caminos entre $s\ and\ all\ t_j \in \mathcal{N}$ $\forall\ s \neq t_j$	Caminos entre $s_i\ and\ all\ t_j \in \mathcal{N}$ $\forall\ s_i \neq t_j$
MDPAlg (DN)	-	$O(N^2)$	$O(N^2)$	$O(N^3)$
MDPAlg (DE)	-	$O(N^2)$	$O(N^3)$	$O(N^4)$
Dijkstra (DN y DE)	$O(N^2)$ [Cormen et al., 2001]	$O(N^3)$	$O(N^4)$	$O(N^5)$
Dijkstra-Cola de prioridad (DN y DE)	$O(\frac{N^2}{2} \cdot log(N))$ [Cormen et al., 2001]	$O(N^3)$	$O(N^4)$	$O(N^5)$
Dijkstra-Fibonacci (DN y DE)	$O(N \cdot log(N) + \frac{N^2}{2})$ [Cormen et al., 2001]	$O(N^3)$	$O(N^4)$	$O(N^5)$
Robertson (DN)	-	$O(N^3)$ [Robertson & Seymour, 1995]	$O(N^4)$	$O(N^5)$
Kawarabayashi (DN)	-	$O(N^2)$ [Kawarabayashi et al., 2012]	$O(N^3)$	$O(N^4)$
Karaata (DN)	$O(N^2)$ [Karaata, 2019]	$O(N^3)$	$O(N^4)$	$O(N^5)$
Eppstein (SD)	-	-	$O(\frac{N^2}{2} + N \cdot log(N) + \frac{N^2}{2} \cdot N)$ [Eppstein, 1998]	$O(N^4)$

caso de MDPAlg, con una única ejecución es capaz de obtener múltiples caminos disjuntos entre un par de nodos, o entre un nodo origen y el resto de nodos de la topología, por eso su primera columna aparece vacía. Independientemente de la implementación de Dijkstra utilizada, MDPAlg, en el peor de los casos, mejora la complejidad computacional en uno o en dos órdenes de magnitud, en función del tipo de caminos descubiertos. Lo mismo ocurre con el resto de propuestas, donde MDPAlg al menos reduce un orden de magnitud la complejidad computacional con respecto a sus competidores. Además, incluso propuestas como la de Eppstein, que se centran simplemente en la búsqueda de múltiples caminos, sin añadir el factor adicional de disjuntividad, solamente igualan a MDPAlg. Por lo tanto, MDPAlg se coloca como un algoritmo capaz de obtener múltiples caminos disjuntos entre múltiples pares de nodos que reduce la complejidad computacional del problema de la búsqueda de caminos disjuntos con respecto a sus competidores.

6.3 Evaluación

Este apartado se centra en evaluar MDPAlg midiendo la cantidad de caminos disjuntos descubiertos y el tiempo de convergencia que le lleva descubrir dichos caminos. Además,

también se incluye una evaluación cuantitativa de la complejidad computacional, en la que se mide la evolución que sufre el tiempo de convergencia conforme se aumenta el tamaño de la red, que sirve como método de validación del estudio teórico de la complejidad computacional realizado en este capítulo. Con estas tres medidas, complejidad computacional, número de caminos disjuntos descubiertos y tiempo de convergencia, se puede cuantificar de manera objetiva la calidad de MDPAlg, y analizar su viabilidad como solución alternativa a las propuestas centralizadas estudiadas durante el capítulo del estado del arte.

Para la comparativa de resultados se ha elegido el algoritmo de Dijkstra, ya que es un algoritmo cuya eficiencia está más que contrastada debido a su uso a lo largo de los años, que además se utiliza como algoritmo de referencia tanto en la fase de análisis de costes de MDPAlg, como en muchos otros trabajos dedicados a la búsqueda de múltiples caminos disjuntos. Asimismo, la elección del algoritmo de Dijkstra también es útil para validar el estudio teórico de la complejidad computacional, puesto que se realizó el estudio teórico de la complejidad computacional para este algoritmo. Por lo tanto, si los valores teóricos y los experimentales de la complejidad computacional coinciden, se puede dar por válido el estudio teórico realizado en el Apartado 6.2. No obstante, los resultados experimentales pueden mostrar una ligera desviación ascendente con respecto a los datos teóricos, fruto de agentes externos a la plataforma de evaluación, ya que ésta se ejecuta en un sistema operativo que tiene más procesos en ejecución trabajando de forma concurrente.

La herramienta de cálculo Matlab se vuelve a elegir como la plataforma de análisis en la que realizar la evaluación por varios motivos. En primer lugar, Matlab proporciona un conjunto de instrumentos intuitivos y de fácil manejo muy útiles para procesar y representar datos de forma masiva. En segundo lugar, Matlab cuenta con la librería de Wang [Wang, 2004], una implementación en Matlab del algoritmo de Dijkstra utilizada en el capítulo anterior, por lo que parte del trabajo estaba previamente desarrollado. Además, Matlab también proporciona un entorno de programación y depuración ágil en el que se acortan los tiempos de desarrollo en comparación con lenguajes de programación clásicos como C o Java.

En relación a las topologías seleccionadas para realizar el proceso de evaluación, se vuelven a utilizar las mismas que en el capítulo anterior, solo que se sustituye la topología regular simple, por el grafo extremo (G_E) utilizado en la Sección 6.2, para validar el estudio teórico de complejidad computacional. Por lo tanto, se utilizará G_E, la malla regular bidimensional, y las topologías aleatorias con los modelos de conexión de Barabási-Albert y Waxman. Además, al igual que en el capítulo anterior, el tamaño de la red se incrementará de forma lineal en los tres tipos de topologías, partiendo de un mínimo de cuatro nodos en la malla bidimensional, un mínimo de diez nodos en las topologías aleatorias, y un mínimo de veinte nodos en G_E; hasta llegar a un máximo de cien nodos en todas ellas. Además, al utilizarse las topologías del capítulo anterior, el retardo de propagación de los enlaces varía entre 0 y 1,5ms. No obstante, se han transformado estos

valores de retardo en un coste de enlace, sin magnitud asociada, que varía entre 0 y 1500, dado que estos algoritmos son agnósticos a la aplicación final de los caminos descubiertos, evitando con esta adaptación que los algoritmos trabajen con números decimales.

Por lo tanto, el apartado de resultados se dividirá en tres epígrafes, una primera sección que muestra los resultados de la complejidad computacional, y un segundo y tercer epígrafes que reflejan el tiempo de convergencia y el número de caminos disjuntos descubiertos, agrupando así el conjunto de resultados en tres bloques en función de la métrica. Esta división facilita la validación del estudio teórico de la complejidad computacional, al dedicarle en exclusiva un apartado a la complejidad computacional.

Todos los experimentos se han ejecutado sobre la misma plataforma hardware, un ordenador con sistema operativo Linux que cuenta con un procesador Intel(R) Core(TM) i7-8700K CPU, 32 GB RAM y disco duro de estado sólido de 256GB. Además, cada experimento se repite diez veces y se calculan los valores medios de los diez lanzamientos y los intervalos de confianza al 95 %. La Tabla 6.2 resume el proceso de evaluación realizado, para tener agrupado el conjunto de pruebas que se realizan.

6.3.1 Resultados

Como se ha explicado anteriormente, esta sección evalúa la complejidad computacional, el tiempo de convergencia y el número de caminos descubiertos en tres apartados distintos centrándose, cada uno de ellos, en una de las métricas.

Complejidad computacional

La evaluación de la complejidad computacional que presentan, tanto MDPAlg como el algoritmo de Dijsktra, se ha llevado a cabo en varias fases. En primer lugar, se quiere validar el estudio teórico de la complejidad computacional realizado en la Sección 6.2, por lo que se han replicado las condiciones de contorno del estudio teórico. Estas condiciones consisten en llevar al extremo a los algoritmos de búsqueda de caminos disjuntos, obteniendo todos los caminos disjuntos posibles entre todos los pares de nodos existentes en G_E. Además, se ha anotado el tiempo que emplea cada algoritmo en obtener todos los caminos disjuntos, variando el tamaño de G_E para establecer una relación entre el tiempo invertido en obtener los caminos disjuntos y el número de nodos de la red. Esta relación es en realidad la complejidad computacional, un estimador que muestra la tendencia o

Tabla 6.2: Resumen del escenario de evaluación

Topología	**Coste enlaces**	**Tamaño de la red (nodos)**	**Conectividad**	**Número de ejecuciones**	**Medidas**
G_E	0 a 1500	20 a 100	19 a 99	10	Complejidad computacional, número de caminos disjuntos, y tiempo de convergencia
Malla bidimensional	0 a 1500	4 a 100	2 a 3.6		
Aleatorias	0 a 1500	10 a 100	2, 4, 6		

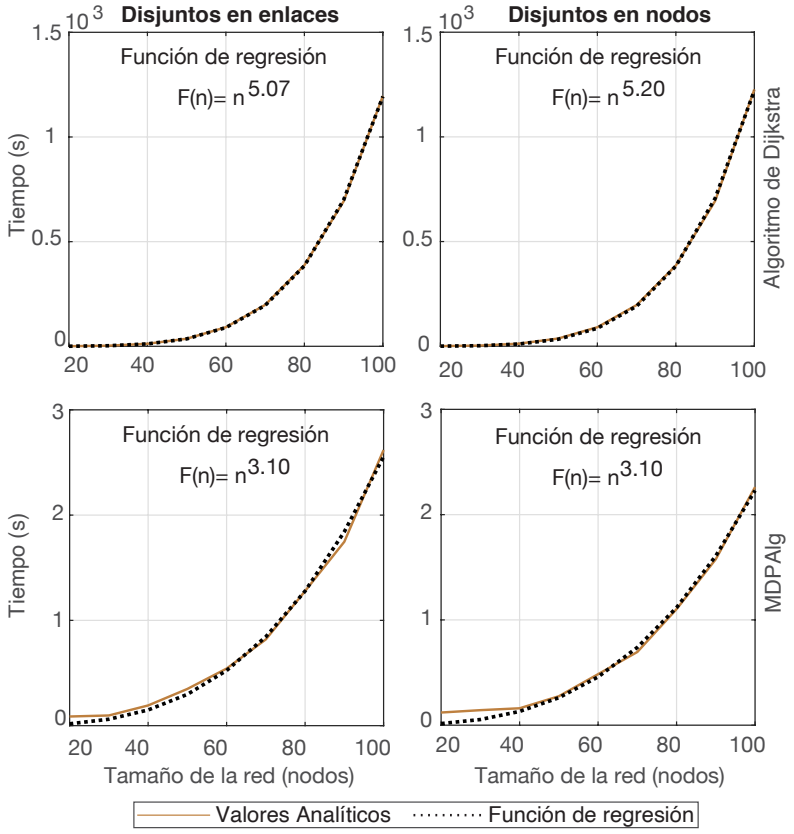


Figura 6.4: Complejidad computacional en G_E. Fuente: elaboración propia

evolución del esfuerzo temporal que tiene que hacer el algoritmo para resolver el problema. El mismo proceso se puede realizar para estimar la complejidad computacional en relación a la cantidad de memoria consumida por los protocolos. Sin embargo, esta medida no presenta interés, puesto que el tamaño de las estructuras de datos empleadas por el algoritmo de Dijkstra y por MDPAlg son insignificantes con respecto a la cantidad de memoria RAM que tienen los ordenadores actuales (por encima de los 8GB). En el peor de los casos analizados, la topología con cien nodos, el algoritmo MDPAlg utiliza una matriz de dimensión 100 celdas x 100 celdas, con un tamaño de 32 *bits* por celda, lo que supone un total de 320kb de datos (40kB), una cantidad de memoria irrelevante para los ordenadores actuales, lo que demuestra que el análisis del consumo de memoria no reviste interés. Por lo tanto, para medir la complejidad computacional, solamente se va a valorar el tiempo invertido en calcular todos los caminos disjuntos, ya que el tiempo de demora en obtener la solución sí que es una medida crítica. Una vez que se ha evaluado la complejidad computacional en G_E, el procedimiento se repite en el resto de topologías, la malla regular y las topologías aleatorias, midiendo el tiempo invertido en obtener todos los caminos disjuntos entre todos los pares de nodos, y representando su evolución en función del tamaño de la red.

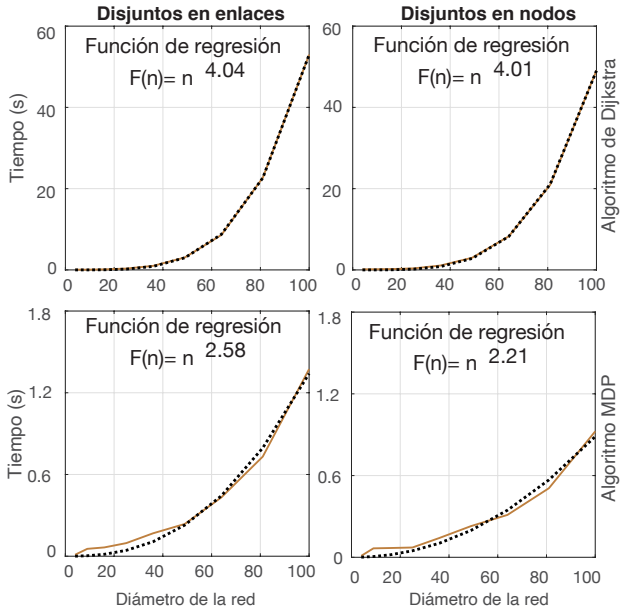
La Figura 6.4 muestra la complejidad computacional resultante de ejecutar el algoritmo de Dijkstra y MDPAlg en G_E para obtener todos los caminos disjuntos entre todos los pares de nodos. La mitad izquierda de la figura muestra los datos del modo caminos

disjuntos en enlaces, y el lado derecho los del modo disjunto en nodos. Asimismo, la mitad superior recoge los valores para el algoritmo de Dijkstra, mientras que la mitad inferior lo hace para MDPAlg. Además, la figura recopila el tiempo que invierte cada algoritmo en obtener el conjunto de caminos disjuntos, medido de forma analítica, y la función de regresión asociada a dichos valores. Esta función se obtiene mediante un análisis de regresión, un método estadístico que relaciona conjuntos de datos entre los que existe una conexión, permitiendo establecer una relación directa entre el tiempo invertido en obtener los caminos disjuntos y el tamaño de la red. En este caso en particular, el tiempo invertido se corresponde con la variable dependiente, y el tamaño de la red con la variable independiente. Matlab incluye este análisis de regresión en su herramienta *Curve Fitting Toolbox*, que obtiene la relación entre grupos de datos en forma de función, de manera rápida e intuitiva. La curva de regresión obtenida tras realizar este proceso, se muestra en la Figura 6.4 en forma de línea punteada, mientras que la función que la representa queda recogida en la parte superior de cada gráfica. Como se puede apreciar, el método es bastante eficaz, obteniendo una función de regresión que se ajusta con precisión a los datos recogidos analíticamente. De la función de regresión, interesa el exponente, ya que éste está relacionado de forma directa con los resultados obtenidos en el estudio teórico de la complejidad computacional. En dicho estudio se devolvían los resultados en la forma $O(N^z)$, que equivale a una función de regresión $fr(N) = N^z$, donde N representa el tamaño de la red.

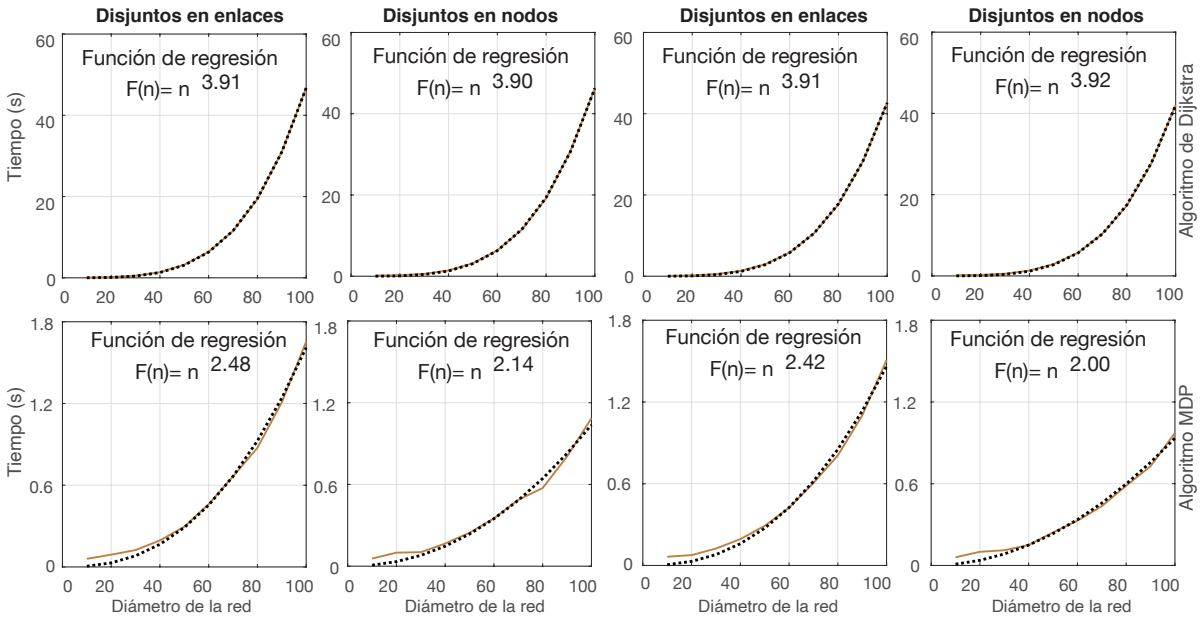
En vista de los resultados, la complejidad computacional del algoritmo de Dijkstra, obtenida de forma experimental, se ajusta a los resultados del estudio teórico, proporcionando en ambos casos una complejidad computacional que evoluciona exponencialmente en una potencia quinta. No obstante, la complejidad computacional obtenida experimentalmente es ligeramente superior a la calculada de forma teórica, debido a la ejecución simultánea de procesos externos por parte del sistema operativo. Por lo tanto, dada la afinidad entre los resultados teóricos y los experimentales, el estudio teórico de la complejidad computacional para el algoritmo de Dijkstra queda validado.

En cuanto a los resultados experimentales de MDPAlg, en ambos modos (caminos disjuntos en enlaces y caminos disjuntos en nodos) se obtiene la misma complejidad computacional, que se corresponde con una función exponencial de orden tres. Sin embargo, el estudio teórico arrojaba una función exponencial $O(N^4)$ para el modo disjunto en enlaces, y una función exponencial ($O(N^3)$ para el modo disjunto en nodos. Esta diferencia entre el estudio teórico y los datos experimentales se debe al alto grado de conectividad entre vecinos que presenta G_E. Como todos los nodos están conectados entre sí, los caminos calculados tienden a elegir los caminos con menor coste y menor número de saltos, por lo que se tiende a seleccionar el enlace directo que une a los nodos origen y destino para el primer camino, y el resto utilizan un tercer nodo como punto de conexión entre el origen y destino, dando lugar a caminos de dos saltos. Por lo tanto, los caminos descubiertos entre un par de nodos no alcanzan el peor caso, utilizar todos los enlaces de

la topología, sino que utilizan solamente un subconjunto del mismo, de ahí que se reduzca la complejidad computacional del modo caminos disjuntos en enlaces. Pero eso no es todo, sino que la solución tiende a ser óptima, ya que al existir una conectividad tan alta, los caminos seleccionados, con un único salto a un nodo vecino, son capaces de comunicar los nodos origen y destino, lo que evita que se compartan nodos entre caminos disjuntos, cuyo efecto visible es que los caminos obtenidos son disjuntos en nodos. Este hecho es perfectamente posible ya que por definición los caminos disjuntos en nodos son también disjuntos en enlaces. De hecho, algunas soluciones vistas en el estado del arte, como



a) Malla rectangular.



b) Aleatoria modelo Waxman. c) Aleatoria modelo Barabási-Albert.

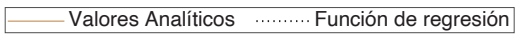


Figura 6.5: Complejidad computacional en el resto de topologías.

[Suurballe, 1974] o [Bhandari, 1994], a pesar de calcular caminos disjuntos en enlaces, pueden obtener caminos disjuntos en nodos en función de la topología, como se vio en el ejemplo de la Figura 2.4. Por lo tanto, el estudio teórico de la complejidad computacional de MDPAlg también queda validado, ya que existe una afinidad de resultados en el modo caminos disjuntos en nodos, y las diferencias existentes en el modo caminos disjuntos en enlace quedan justificadas.

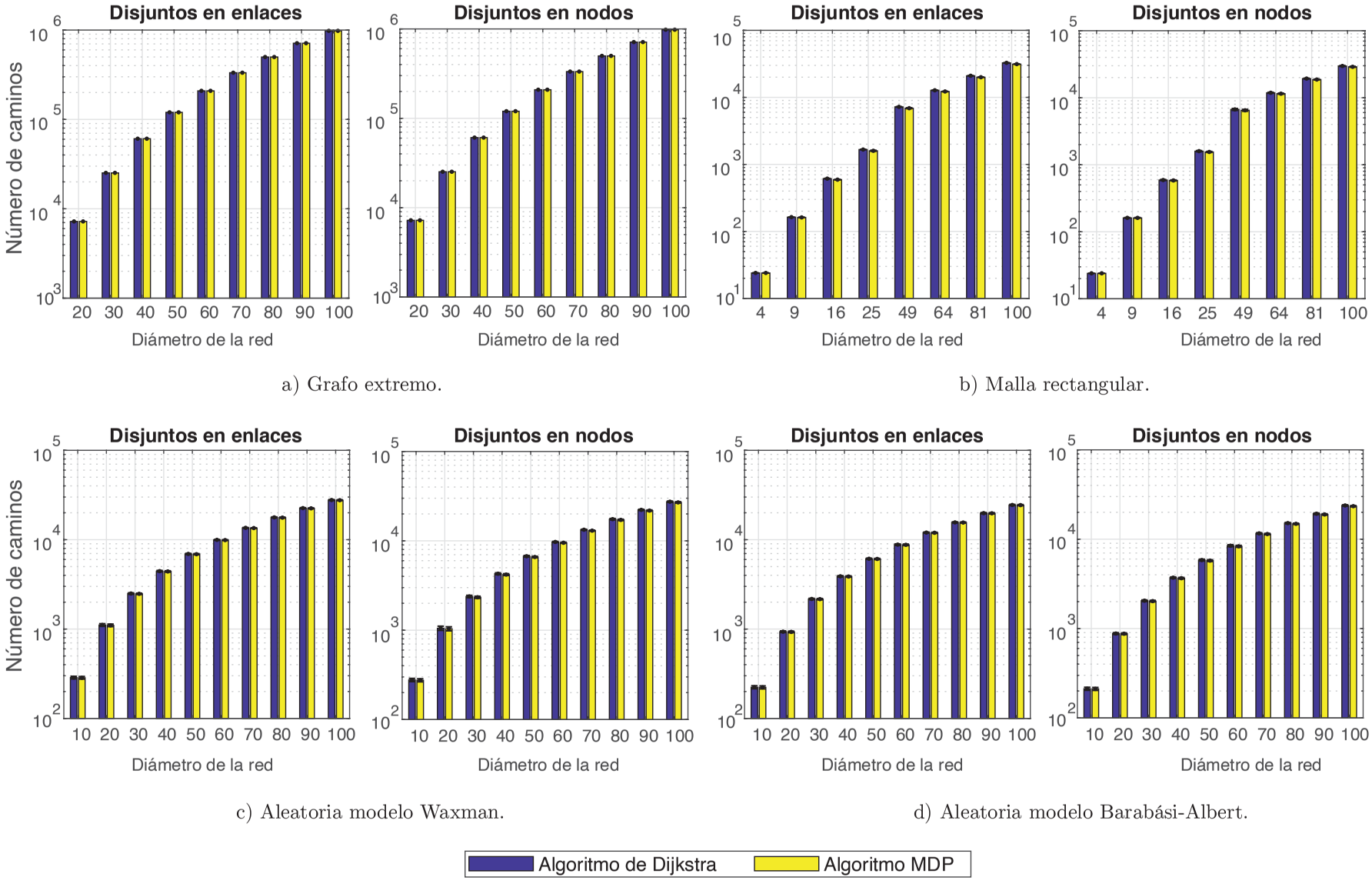
Para el resto de topologías se ha seguido el mismo proceso, se han calculado todos los caminos disjuntos entre todos los pares de nodos en ambos modos de operación. Los resultados quedan recogidos en la Figura 6.5, que proporciona en la parte superior la complejidad computacional resultante en la malla regular, y en la parte inferior los resultados para las topologías aleatorias con un grado de conectividad 4. Se han omitido los resultados para las topologías aleatorias con grados de conectividad 2 y 6 debido a la similitud en los resultados. Independientemente de la topología, MDPAlg reduce en uno o dos niveles el exponente de la función de regresión asociada a la complejidad computacional con respecto a los resultados arrojados por el algoritmo de Dijkstra.

MDPAlg no solo reduce la complejidad computacional, sino que también reduce drásticamente el tiempo invertido en calcular los caminos disjuntos, como se verá en el apartado dedicado al tiempo de convergencia.

Número de caminos

Este apartado cuantifica el número de caminos disjuntos que obtienen el algoritmo de Dijkstra y MDPAlg cuando se intentan calcular todos los caminos disjuntos posibles entre todos los pares de nodos. Las topologías utilizadas son las mismas que en el apartado anterior, G_E, la malla bidimensional, y las topologías aleatorias con los modelos de Barabási-Albert y Waxman. Al igual que en el apartado anterior, dada la similitud entre los resultados, solo se muestran los correspondientes a las topologías aleatorias con un grado de conectividad cuatro. La Figura 6.6 recoge los resultados de este análisis, mostrando en cuatro gráficas el número de caminos obtenidos por ambos protocolos en cada topología. Además, en cada subfigura, se muestran los resultados para el modo disjunto en enlaces y para el modo disjunto en nodos. El comportamiento mostrado por las gráficas es similar en todas las topologías, el número de caminos descubiertos se incrementa conforme aumenta el tamaño de la red. Esta conclusión es obvia, ya que al contener la topología un mayor número de nodos, se incrementa el número de pares de nodos origen-destino entre los que calcular múltiples caminos disjuntos. Además, al incrementar el tamaño de la red también se incrementa la probabilidad de obtener más caminos disjuntos entre un par de nodos, ya que existen nuevas vías con potencial para descubrir nuevos caminos disjuntos.

Otro comportamiento que se repite es la similitud de resultados entre el algoritmo de Dijkstra y MDPAlg, independientemente de la topología y el tamaño de la red. En todos los casos, MDPAlg bien iguala al algoritmo de Dijkstra, bien obtiene un rendimiento ligeramente inferior, que en el peor de los casos no supera el 12 % del conjunto total de caminos descubiertos (resultado similar el que mostraba 1S-MDP con respecto a Dijkstra). Sin embargo, esta pequeña merma se ve más que compensada por la complejidad computacional que obtiene MDPAlg, que reduce en uno o dos órdenes de magnitud la complejidad computacional del algoritmo de Dijkstra. Ambos efectos se deben al proceso único de análisis de costes llevado a cabo por MDPAlg, que obtiene múltiples caminos disjuntos entre múltiples pares de nodos a costa de recopilar más información topológica. No obstante, este proceso único de análisis de costes obtiene menor precisión que el algoritmo de Dijkstra, ya que la información se recolecta una única vez y es utilizada por varios nodos destino para construir los caminos disjuntos, mientras que el algoritmo de Dijkstra realiza un análisis de costes por cada camino disjunto descubierto entre un par de nodos, lo que aumenta el número de operaciones y la complejidad computacional.



a) Grafo extremo.

b) Malla rectangular.

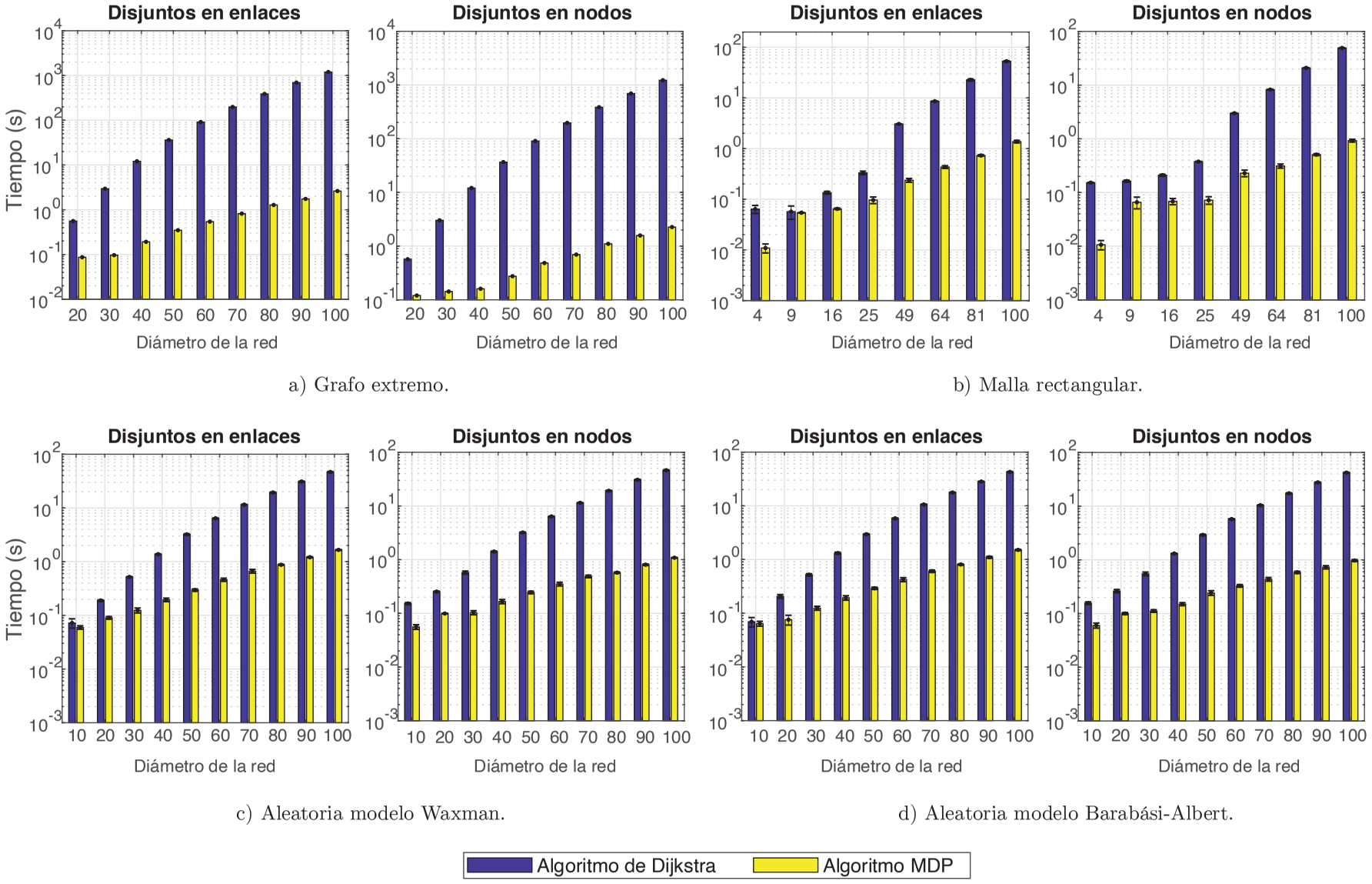
c) Aleatoria modelo Waxman.

d) Aleatoria modelo Barabási-Albert.

Figura 6.6: Caminos descubiertos.

Tiempo de convergencia

El tercer apartado evalúa el tiempo de convergencia bajo la misma premisa que los dos apartados anteriores, obtener todos los caminos disjuntos posibles entre todos los pares de nodos. Los resultados se muestran en una única figura, la Figura 6.7, que presenta el tiempo de convergencia obtenido en cuatro modelos topológicos con distintos tamaños de red. Los resultados confirman lo que dejaba intuir el apartado dedicado a la complejidad computacional, MDPAlg reduce drásticamente el tiempo de convergencia con respecto al algoritmo de Dijkstra. En todas las topologías y tamaños de red, el tiempo de convergencia de MDPAlg se sitúa por debajo del tiempo de convergencia del algoritmo de Dijkstra. Esta diferencia se incrementa cuanto mayores son el tamaño y la conectividad de la red, ya que MDPAlg tiene que realizar menos cálculos para obtener, prácticamente, los mismos caminos disjuntos que el algoritmo de Dijkstra. Los buenos resultados obtenidos por MDPAlg se deben, principalmente, a la simplificación de la operativa matemática, que consigue recopilar suficiente información para construir múltiples caminos disjuntos entre múltiples pares de nodos con un único proceso de análisis de costes del grafo topológico. Propuestas alternativas como el algoritmo de Dijkstra necesitan realizar varios análisis de costes del grafo topológico para obtener caminos disjuntos entre un par de nodos, lo que incrementa el número de operaciones matemáticas, el tiempo de convergencia y la complejidad computacional. Además, MDPAlg, en las situaciones más extremas (G_E con tamaños de red de cien nodos), reduce tres órdenes de magnitud el tiempo de convergencia con respecto al que obtiene Dijkstra. Esto se traduce en que MDPAlg obtiene todos los caminos disjuntos en un tiempo, asumible, que ronda el segundo aproximadamente, mientras que el algoritmo de Dijkstra necesita 1000 segundos para llegar a un resultado similar. En el resto de topologías, con grandes tamaños de red, esta reducción se sitúa en torno a los dos órdenes de magnitud, decremento suficientemente representativo como para compensar la diferencia de caminos obtenidos. En tamaños de red inferiores, el algoritmo de Dijkstra llega a superar el umbral del segundo con topologías de entre treinta y cincuenta nodos, mientras que MDPAlg alcanza al umbral del segundo a partir de los ochenta nodos, lo que confirma la escalabilidad de MDPAlg.



a) Grafo extremo.

b) Malla rectangular.

c) Aleatoria modelo Waxman.

d) Aleatoria modelo Barabási-Albert.

Figura 6.7: Tiempo de convergencia.

Capítulo 7

Conclusiones y trabajos futuros

Este último capítulo recopila las principales contribuciones de la Tesis Doctoral, detalladas en los Capítulos 4, 5, 6, así como las conclusiones fundamentales a las que se ha llegado tras el trabajo de estos últimos años. Además, este capítulo también recoge líneas futuras sobre las que se puede trabajar en los próximos años, utilizando los resultados obtenidos por la presente Tesis. Por lo tanto, este último epígrafe resume brevemente los logros alcanzados tras años de trabajo, los cuales pueden servir de inspiración a otros investigadores.

7.1 Amaru

Amaru se presenta como un protocolo de encaminamiento In-Band, diseñado para ser autoconfigurable, escalable y rápido en la restauración de rutas ante fallos de red, que permite comunicar las capas de control e infraestructura en redes SDN de forma distribuida, reaprovechando el equipamiento disponible en la capa de infraestructura. Además, Amaru introduce un enfoque distinto al utilizado en las propuestas de protocolos In-Band, al utilizar técnicas de exploración de red en lugar de emplear mecanismos basados en el estado del enlace. Por todo ello, Amaru consigue reducir drásticamente el tiempo de convergencia y el número de paquetes utilizados, al mismo tiempo que garantiza la estabilidad de la red al asegurar la restauración de caminos en menos de 50ms.

Amaru incluye un sencillo mecanismo de propagación de etiquetas que explora la red de forma distribuida, permitiendo establecer múltiples canales de comunicación entre el controlador y cada uno de los switches que están gobernados bajo su dominio, usando la información que contienen las etiquetas. Este mecanismo está parametrizado con el objetivo de que Amaru sea un protocolo flexible que pueda adaptarse a distintas situaciones mediante tres variables: N, L y P. La variable N restringe el número máximo de etiquetas/caminos que puede aprender cada switch, el parámetro L indica el número de elementos en común (enlaces) que pueden tener los caminos entre sí, y la variable P limita el número máximo de saltos que pueden tener los caminos. Además, gracias a la combinación de las técnicas de exploración de red, junto con el sistema de etiquetado,

Amaru permite seleccionar, en cada switch, el canal de comunicación con el controlador basándose en dos criterios: el número de saltos que hay hasta el controlador, si atiende al número de elementos que contiene la etiqueta AMAC, o en función de la latencia, si selecciona la primera AMAC aprendida. Esta última característica aumenta aún más la flexibilidad de Amaru, ya que además de poder definir las características de los caminos, permite seleccionarlos basándose en dos criterios: número de saltos o latencia; característica muy poco común en los protocolos de red.

Con respecto a la evaluación, Amaru se ha implementado en tres plataformas distintas, dos de ellas basadas en simulación (Python y OMNeT++), y la tercera en emulación, obteniendo resultados prometedores. Amaru descubre múltiples caminos entre un switch raíz y el resto de switches de la red sin penalizar la escalabilidad, ya que reduce el tiempo de convergencia en al menos tres órdenes de magnitud con un consumo de paquetes contenido que puede llegar a ser hasta 30 veces menor que las propuestas de [Omizo et al., 2016, Sakic et al., 2020]. Además, también reduce el tiempo de recuperación ante fallos, situándolo por debajo del umbral de 50 ms que se fija como tiempo máximo para recuperar un canal, y mejorando en torno a un 10 % la propuesta de [Asadujjaman et al., 2018].

La falta de un estándar por parte de organismos de estandarización como el IEEE, la ONF, o el ITU, en los protocolos de encaminamiento In-Band en redes SDN, hacen que Amaru sea un candidato perfecto para ello, ya que proporciona un protocolo sencillo, autoconfigurable, con una escalabilidad y tiempos de recuperación contrastados, que únicamente necesita un agente Amaru instalado en cada uno de los switches. Por estos motivos Amaru tiene potencial para implantarse de forma oficial en las redes SDN In-Band, al no existir una alternativa que cumpla con todos sus requisitos.

7.2 Protocolos de descubrimiento de caminos múltiples disjuntos

Tanto I-MDP como 1S-MDP se presentan como protocolos orientados a descubrir múltiples caminos disjuntos trabajando de forma distribuida en un entorno meramente centralizado, como son las redes SDN. Para ello, hacen uso del concepto de red SDN híbrida, mediante el cual se modifica la estructura jerárquica de las redes SDN, que fija al controlador como el dispositivo responsable de gobernar a la capa de infraestructura que hay bajo su dominio. Esta hibridación delega en la capa de infraestructura funciones que típicamente realizaba el controlador, siendo ahora los switches de la red los encargados de realizar, de manera independiente, funciones de red sin la intervención del controlador. Al finalizar la operación, la capa de infraestructura devuelve el resultado al controlador, para que éste pueda utilizar dicha información. De esta forma, se reduce la sobrecarga de cómputo en el controlador, ya que ciertas tareas las realiza ahora la capa de infraestructura

de forma distribuida, lo que permite aprovechar las ventajas que aporta un sistema centralizado junto con las bondades que brindan los sistemas distribuidos.

Ambos protocolos, al igual que Amaru, se basan en técnicas de exploración de red, solo que prescinden del encaminamiento jerárquico utilizado por Amaru, sustituyéndolo por un sistema de etiquetado local que no tiene en cuenta información de etapas previas, sino que únicamente recopila información basándose en el orden de llegada de los paquetes de exploración. Además, cuentan con un proceso de búsqueda de caminos disjuntos estructurado en dos fases que trabaja de forma distribuida, cuyo resultado de búsqueda se envía al controlador para que gestione el uso de los caminos. La primera fase se encarga de recoger información topológica aplicando las técnicas de exploración de red (fase de exploración), mediante el uso de una difusión controlada de paquetes por la red, mientras que la segunda fase es la responsable de encontrar los caminos disjuntos basándose en la información recogida en la fase previa (fase de selección). Sin embargo, existen diferencias importantes entre ambos protocolos, ya que 1S-MDP surge como una versión mejorada de I-MDP. Mientras que I-MDP realiza varios procesos de búsqueda entre un par de switches (exploración de la red más selección de camino), 1S-MDP optimiza la fase de exploración y permite, con un único proceso de exploración, descubrir en paralelo múltiples caminos disjuntos entre múltiples pares de switches. Esta optimización consigue reducir de forma drástica el tiempo de convergencia sin apenas penalizar el número de caminos disjuntos descubiertos. Además, ambos protocolos incorporan la capacidad de descubrir tanto caminos disjuntos en nodos, como caminos disjuntos en enlaces, una característica poco común en los protocolos de búsqueda de caminos disjuntos que trabajan de forma distribuida.

Durante las etapas de diseño y desarrollo de 1S-MDP se introdujeron dos mejoras que incrementan significativamente el número de caminos disjuntos descubiertos, el salto atrás y la modificación de prioridades. El salto atrás es un sistema que permite retroceder a etapas previas del camino disjunto que está en proceso de construcción cuando se llega a un punto muerto en que no hay salida posible. Este retroceso permite rastrear rutas alternativas desde etapas previas al punto muerto que permiten finalizar satisfactoriamente el proceso de construcción del camino disjunto. El sistema de modificación de prioridades modifica ligeramente el orden en el que se almacena la información recogida por la fase de exploración, "penalizando" las entradas que cumplen ciertos requisitos. Posteriormente, en la fase de selección de caminos, se hace uso de esta ordenación especial aplicando técnicas de prohibición de giros, que impiden utilizar entradas "penalizadas" mientras queden entradas sin penalizar libres. La versión final de 1S-MDP incluye ambas mejoras, puesto que su mejoría queda reflejada en el apartado de resultados.

El proceso de evaluación contaba con la experiencia previa de Amaru, por lo que se optó por utilizar únicamente un simulador de eventos discretos que utilizase una librería de un software switch SDN real. Esta combinación confiere la posibilidad de realizar

experimentos a gran escala con un equipamiento hardware modesto, al mismo tiempo que se utiliza un switch SDN muy cercano a un modelo real. El conjunto de pruebas consistió en buscar todos los caminos disjuntos posibles en varios modelos topológicos y medir el número de caminos disjuntos obtenidos y el tiempo que se ha tardado en obtenerlos. Como elemento de comparación se eligió el algoritmo de Dijkstra, ya que es un algoritmo de referencia a través del cual es posible obtener tanto caminos disjuntos en nodos, como caminos disjuntos en enlaces. Además, al tratarse de un algoritmo ampliamente utilizado en el estado del arte para construir propuestas de búsqueda de caminos disjuntos, es un buen elemento de comparación sobre el que sacar conclusiones. Los resultados dan como vencedor a 1S-MDP, puesto que garantiza la escalabilidad en redes de gran tamaño al reducir drásticamente el tiempo de convergencia (entre dos y tres órdenes de magnitud) con respecto al algoritmo de Dijkstra sin apenas penalizar el número de caminos disjuntos descubiertos (variaciones entre el 2 % y el 12 %).

Por lo tanto, con propuestas como I-MDP 1S-MDP se impulsa el crecimiento de las redes híbridas, ya que quedan latente las ventajas que aportan los sistemas híbridos frente a sistemas que apuestan por un modelo completamente centralizado, o un modelo absolutamente distribuido.

7.3 Algoritmo de búsqueda de caminos disjuntos

Debido a los buenos resultados obtenidos por 1S-MDP en entornos híbridos, se plantea la idea de adaptar dicho protocolo a entornos centralizados para aplicarlo en otras disciplinas en las que se necesita un enfoque centralizado. El resultado de esta adaptación es MDPAlg, un algoritmo centrado en la búsqueda de caminos disjuntos que trabaja sobre una representación de la red (grafo topológico), en lugar de operar directamente con los elementos involucrados en la búsqueda de caminos disjuntos, como hacía 1S-MDP. Este grafo topológico es agnóstico, por lo que no especifica el tipo de elementos (nodos) que componen dicho grafo, sino que tan solo establece las conexiones entre dichos elementos y el coste que tiene atravesarlos. Al trabajar con grafos agnósticos, la aplicabilidad de MDPAlg se incrementa, ya que puede trabajar con cualquier aplicación que necesite obtener caminos disjuntos, como por ejemplo tráfico vehícular, automatización de procesos industriales, procesos de evacuación, planificación de rutas de reparto, etc., con tan solo proporcionarle a MDPAlg un grafo que relaciones los elementos entre los que obtener los caminos disjuntos.

MDPAlg, al estar inspirado en 1S-MDP, también estructura su mecanismo de búsqueda de caminos disjuntos en un proceso dividido en dos fases. La primera de ellas, recopila información topológica realizando un análisis de costes sobre el grafo proporcionado, calculando el coste que tiene atravesar la red desde un nodo origen hasta el resto de nodos por cualquier enlace. Este proceso recopila información adicional con respecto a

otras propuestas, que únicamente se centran en recoger información sobre el camino de menor coste hasta cada nodo. Es esta información adicional la que permite construir posteriormente, en la segunda fase, múltiples caminos disjuntos entre el nodo origen y el resto de nodos del grafo. Por lo tanto, al recopilar información adicional, MDPAlg es capaz de obtener múltiples caminos disjuntos entre múltiples pares de nodos analizando una única vez el grafo topológico, lo que reduce significativamente el número de operaciones matemáticas, y por ende, la complejidad computacional.

Para corroborar la drástica reducción de la complejidad computacional, se ha realizado un estudio teórico que analiza la complejidad computacional obtenida por MDPAlg en un escenario extremo, y la compara con la complejidad computacional del algoritmo de Dijkstra. Una vez más, esta comparación no es casual, ya que el algoritmo de Dijkstra es la base sobre la que se construyen numerosos algoritmos que tienen por objetivo calcular caminos disjuntos, por lo que dicho algoritmo constituye un buen punto de referencia con el que comparar MDPAlg. Además, el estudio incluye, de forma resumida, la complejidad computacional que obtienen otras propuestas existentes en la literatura para resolver el mismo problema. Del resultado del estudio se concluye que MDPAlg es más eficiente que sus competidores, ya que obtiene una complejidad computacional menor que el resto de propuestas.

La evaluación de MDPAlg por un lado pretende validar el estudio teórico, y por otro, determinar la calidad de MDPAlg, cuantificando el tiempo de convergencia y el número de caminos disjuntos descubiertos. Para validar de forma experimental el estudio teórico, se han replicado las condiciones de contorno de dicho estudio y se ha medido el tiempo que invierten MDPAlg y el algoritmo de Dijkstra en obtener todos los caminos disjuntos en el mismo modelo topológico, aumentando el tamaño de la red. La curva que proporciona la evolución de dicho tiempo en función del tamaño de la red constituye la complejidad computacional, cuya función coincide con los resultados obtenidos en el estudio teórico. Además, para corroborar los resultados, se ha extendido la evaluación a otras topologías, mostrando en todos los casos una reducción del exponente de la función que describe la complejidad computacional entre uno y tres órdenes por parte de MDPAlg con respecto al algoritmo de Dijkstra. En cuanto a la calidad de los resultados, independientemente del modelo topológico, MDPAlg disminuye entre uno y tres órdenes de magnitud el tiempo de convergencia al mismo tiempo que mantiene el número de caminos descubiertos con una oscilación menor del 12 % con respecto al algoritmo de Dijkstra, obteniendo en conjunto una solución escalable que trabaja con soltura en redes grandes.

Todas estas características hacen de MDPAlg un algoritmo versátil, capaz de adaptarse a distintos entornos que requieran utilizar múltiples caminos disjuntos (redes, tráfico vehícular, robótica, ...) mediante un proceso escalable que obtiene los múltiples caminos disjuntos de forma rápida, minimizando el tiempo de espera. Todo ello se ha conseguido gracias al estudio previo realizado en redes SDN híbridas, trasladando los conceptos aprendidos a un entorno completamente centralizado.

7.4 Trabajos Futuros

Las líneas de investigación que abre la presente Tesis Doctoral están relacionadas con el encaminamiento In-Band en redes SDN, el concepto de red SDN híbrida, y la aplicabilidad de las soluciones multicamino.

En cuanto al encaminamiento In-Band, la falta de un estándar por parte de organismos oficiales hace que se pueda introducir Amaru en organizaciones como la ONF dándolo a conocer oficialmente a la comunidad SDN, y promover así su uso por parte del colectivo SDN, de forma que se termine convirtiendo en un referente del encaminamiento In-Band. Además, se puede estudiar la implementación de Amaru en switches software más sofisticados, como OVS, o ser aún más ambiciosos e implementar Amaru en P4, un lenguaje de alto nivel diseñado específicamente para programar el comportamiento del plano de datos en dispositivos de red, y que es compatible con múltiples plataformas (switches software, ASICs, NICs, FPGAs, etc.). También se puede transformar el protocolo Amaru en un algoritmo que trabaje en entornos centralizados, de forma análoga al proceso de transformación realizado en la presente Tesis Doctoral para transformar 1S-MDP en MDPAlg, lo que ampliaría la aplicabilidad de Amaru a otras disciplinas.

Con respecto a las redes SDN híbridas, éstas aún tienen un largo recorrido por explorar, ya que SDN es un concepto relativamente reciente que está en pleno auge de expansión, y más aún con los retos que introducen las redes de nueva generación, el 5G y el 6G. Estas redes, en su núcleo incluyen SDN, por lo que el controlador será el elemento central que gestione todos los servicios, y tendrá una carga computacional considerable. Por lo tanto, impulsar las redes SDN híbridas creando nuevos protocolos que liberen de carga computacional al controlador será un elemento clave para aumentar la escalabilidad de las redes del futuro. Para ello, todo lo aprendido con el protocolo 1S-MDP es la base de la que partir para desarrollar protocolos en entornos SDN híbridos. En relación a 1S-MDP, para potenciar su uso en redes SDN híbridas, se puede implementar en plataformas reales, bien utilizando lenguajes como P4, bien adaptando la librería de BOFUSS de ns-3 para trabajar en entornos emulados.

En lo que concierne a la aplicabilidad de las soluciones multicamino, se puede optar por varias vías. En primer lugar, evaluar el comportamiento de los caminos múltiples disjuntos que obtienen tanto 1S-MDP como MDPAlg en aplicaciones típicas de redes de comunicaciones (balanceo de carga, caminos de backup, dividir flujos para asegurar la integridad de los datos, etc.), y después comprobar las diferencias que existen entre ejecutarlo en un entorno híbrido con 1S-MDP, o ejecutarlo en el controlador de una red SDN con MDPAlg. Esta comparativa obtendría una visión clara de las ventajas y los inconvenientes que aportan ambos enfoques. Además, se pueden abrir nuevos nichos de investigación con MDPAlg, poniendo en marcha líneas de investigación y colaboraciones entre grupos de investigación de otras disciplinas para problemas que necesiten utilizar múltiples caminos disjuntos.

Bibliografía

[Abe et al., 2015] Abe, J. O., Mantar, H. A., & Yayimli, A. G. (2015). K-Maximally Disjoint Path Routing Algorithms for SDN. In *Proceedings - 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2015* (pp. 499–508).: IEEE.

[Acharya et al., 2020] Acharya, H. B., Hamilton, J., & Shenoy, N. (2020). From spanning trees to meshed trees. In *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)* (pp. 391–395).

[Ahmad, 2002] Ahmad, R. (2002). Routing and wavelength assignment algorithms in IP over DWDM networks. In *IEEE Students Conference, ISCON '02. Proceedings.*, volume 1 (pp. 154–157).

[Al-Fares et al., 2008] Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4), 63–74.

[Al-Turjman & Alturjman, 2020] Al-Turjman, F. & Alturjman, S. (2020). 5G/IoT-enabled UAVs for multimedia delivery in industry-oriented applications. *Multimedia Tools and Applications*, 79(13), 8627–8648.

[Alizadeh et al., 2014] Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V. T., Matus, F., Pan, R., Yadav, N., & Varghese, G. (2014). CONGA: Distributed Congestion-aware Load Balancing for Datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4), 503–514.

[Alsaeedi et al., 2019] Alsaeedi, M., Mohamad, M. M., & Al-Roubaiey, A. A. (2019). Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey. *IEEE Access*, 7, 107346–107379.

[Alvarez-Horcajo et al., 2017] Alvarez-Horcajo, J., Martinez-Yelmo, I., Rojas, E., Carral, J. A., & Lopez-Pajares, D. (2017). New cooperative mechanisms for software defined networks based on hybrid switches. *Transactions on Emerging Telecommunications Technologies*, 28(8), 1–15.

[Asadujjaman et al., 2018] Asadujjaman, A., Rojas, E., Alam, M. S., & Majumdar, S. (2018). Fast Control Channel Recovery for Resilient In-band OpenFlow Networks. In

2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft) (pp. 19–27).

[Azzouni et al., 2017] Azzouni, A., Boutaba, R., Trang, N. T. M., & Pujolle, G. (2017). sOFTDP: Secure and efficient topology discovery protocol for SDN. *arXiv preprint arXiv:1705.04527*.

[Barabási & Albert, 1999] Barabási, A.-L. & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439), 509–512.

[Bentstuen & Flathagen, 2018] Bentstuen, O. I. & Flathagen, J. (2018). On Bootstrapping In-Band Control Channels in Software Defined Networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)* (pp. 1–6).

[Bhandari, 1994] Bhandari, R. (1994). Optimal diverse routing in telecommunication fiber networks. *Proceedings of INFOCOM '94 Conference on Computer Communications*, 3, 1498–1508.

[Brocade, 2012] Brocade (2012). Hybrid OpenFlow, the Brocade Way. `https://blog.ipspace.net/2012/06/hybrid-openflow-brocade-way.html`. Accessed: August 2020.

[Brocade, 2015] Brocade (2015). Brocade FastIron OpenFlow. `https://support.alcadis.nl/Support_files/Ruckus/ICX//Ruckus%20Best%20Practice%20Guide/Brocade%20FastIron%20OpenFlow%20Deployment%20Guide.pdf`. Accessed: August 2020.

[Canini et al., 2017] Canini, M., Salem, I., Schiff, L., Schiller, E. M., & Schmid, S. (2017). A Self-Organizing Distributed and In-Band SDN Control Plane. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (pp. 2656–2657).

[Canini et al., 2018] Canini, M., Salem, I., Schiff, L., Schiller, E. M., & Schmid, S. (2018). Renaissance: A Self-Stabilizing Distributed SDN Control Plane. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (pp. 233–243).

[Chan et al., 2018] Chan, K., Chen, C., Chen, Y., Tsai, Y., Lee, S. S. W., & Wu, C. (2018). Fast Failure Recovery for In-Band Controlled Multi-Controller OpenFlow Networks. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 396–401).

[Chaves et al., 2016] Chaves, L. J., Garcia, I. C., & Madeira, E. R. M. (2016). Ofswitch13: Enhancing ns-3 with openflow 1.3 support. In *Proceedings of the Workshop on ns-3* (pp. 33–40).

[Cidon et al., 1999] Cidon, I., Rom, R., & Shavitt, Y. (1999). Analysis of multi-path routing. *IEEE/ACM Transactions On Networking*, 7(6), 885–896.

[Colorni et al., 1991] Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed optimization by ant colonies. In *First European Conference on Artificial Life* (pp. 134–142).

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms*. MIT Press, second edition.

[De Maesschalck et al., 2003] De Maesschalck, S., Colle, D., Lievens, I., Pickavet, M., Demeester, P., Mauz, C., Jaeger, M., Inkret, R., Mikac, B., & Derkacz, J. (2003). Pan-European optical transport networks: An availability-based comparison. *Photonic Network Communications*, 5(3), 203–225.

[De Pellegrini et al., 2006] De Pellegrini, F., Starobinski, D., Karpovsky, M. G., & Levitin, L. B. (2006). Scalable, distributed cycle-breaking algorithms for gigabit Ethernet backbones. *Journal of Optical Networking*, 5(2), 122–144.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1), 269–271.

[Dong et al., 2017] Dong, G.-S., Shen, J., & Sun, L.-Q. (2017). Fast Failure Recovery in Software-Defined Networks. *Advances in Engineering Research*, 130(5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017)), 1579–1582.

[Doshi & Kamdar, 2018] Doshi, M. & Kamdar, A. (2018). Multi-constraint QoS disjoint multipath routing in SDN. In *Moscow Workshop on Electronic and Networking Technologies, MWENT 2018 - Proceedings* (pp. 1–5).: Institute of Electrical and Electronics Engineers Inc.

[Eilam-Tzoreff, 1998] Eilam-Tzoreff, T. (1998). The disjoint shortest paths problem. *Discrete applied mathematics*, 85(2), 113–138.

[Eppstein, 1998] Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on computing*, 28(2), 652–673.

[ETSI, 2012] ETSI (2012). Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. *Introductory White Paper*, (pp.1̃6).

[Euler, 1741] Euler, L. (1741). Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, (pp. 128–140).

[Fan & Yang, 2020] Fan, W. & Yang, F. (2020). Centralized Trust-Based In-Band Control for SDN Control Channel. *IEEE Access*, 8, 4289–4300.

[Farrington & Andreyev, 2013] Farrington, N. & Andreyev, A. (2013). Facebook's data center network architecture. In *Optical Interconnects Conference, 2013 IEEE* (pp. 49–50).: Citeseer.

[Fernandes et al., 2020] Fernandes, E. L., Rojas, E., Alvarez-Horcajo, J., Kis, Z. L., Sanvito, D., Bonelli, N., Cascone, C., & Rothenberg, C. E. (2020). The road to BOFUSS: The basic OpenFlow userspace software switch. *Journal of Network and Computer Applications*, (pp. 102685).

[Fidler & Einhoff, 2004] Fidler, M. & Einhoff, G. (2004). Routing in turn-prohibition based feed-forward networks. In *International Conference on Research in Networking* (pp. 1168–1179).: Springer.

[Ford et al., 2013] Ford, A., Raiciu, C., Handley, M., & Bonaventure, O. (2013). TCP extensions for multipath operation with multiple addresses. RFC 6824.

[Foss et al., 2011] Foss, S., Korshunov, D., Zachary, S., et al. (2011). *An introduction to heavy-tailed and subexponential distributions*, volume 6. Springer.

[Fredman & Tarjan, 1987] Fredman, M. L. & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3), 596–615.

[Freitas et al., 2020] Freitas, M. S., Rosa, P. F., & de Oliveira Silva, F. (2020). ConForm: In-Band Control Flows Self-establishment with Integrated Topology Discovery to SDN-Based Networks. In *Workshops of the International Conference on Advanced Information Networking and Applications* (pp. 100–109).: Springer International Publishing.

[Geng et al., 2018] Geng, H., Shi, X., Wang, Z., & Yin, X. (2018). A hop-by-hop dynamic distributed multipath routing mechanism for link state network. *Computer Communications*, 116(July), 225–239.

[Goltsmann et al., 2017] Goltsmann, P., Zitterbart, M., Hecker, A., & Bless, R. (2017). *Towards a Resilient In-Band SDN Control Channel*. Technical report, Universität Tübingen, Germany.

[González et al., 2018] González, S., la Oliva, A. D., Bernardos, C. J., & Contreras, L. M. (2018). Towards a Resilient Openflow Channel Through MPTCP. In *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)* (pp. 1–5).

[Goransson et al., 2014] Goransson, P., Black, C., & Culver, T. (2014). Openflow limitations. In *Software Defined Networks: A comprehensive Approach* (pp. 116). Morgan Kaufmann.

[Greenberg et al., 2009] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., & Sengupta, S. (2009). VL2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4), 51–62.

[Guo et al., 2003] Guo, Y., Kuipers, F., & Van Mieghem, P. (2003). Link-disjoint paths for reliable QoS routing. *International Journal of Communication Systems*, 16(9), 779–798.

[Hadid et al., 2017] Hadid, R., Karaata, M. H., & Villain, V. (2017). A Stabilizing Algorithm for Finding Two Node-Disjoint Paths in Arbitrary Networks. *International Journal of Foundations of Computer Science*, 28(4), 411–435.

[Hamner & Samsen, 1988] Hamner, M. C. & Samsen, G. R. (1988). Source routing bridge implementation (LANs). *IEEE Network*, 2(1), 33–36.

[Hark et al., 2017] Hark, R., Rizk, A., Richerzhagen, N., Richerzhagen, B., & Steinmetz, R. (2017). Isolated in-band communication for distributed SDN controllers. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops* (pp. 1–2).

[Holzmann & Zitterbart, 2019] Holzmann, P. & Zitterbart, M. (2019). Izzy: A Distributed Routing Protocol for In-band SDN Control Channel Connectivity. In *Proceedings - 2019 IEEE 44th Local Computer Networks Symposium on Emerging Topics in Networking, LCN Symposium 2019* (pp. 18–25).: Institute of Electrical and Electronics Engineers Inc.

[Hong, 2012] Hong, D. (2012). Medical Image Segmentation Based on Accelerated Dijkstra Algorithm. In *Advances in Intelligent Systems* (pp. 341–348). Springer.

[Hong et al., 2016] Hong, D. K., Ma, Y., Banerjee, S., & Mao, Z. M. (2016). Incremental deployment of SDN in hybrid enterprise and ISP networks. In *Proceedings of the Symposium on SDN Research* (pp. 1–7).

[Hossen et al., 2019] Hossen, M. S., Rahman, M. H., Al-Mustanjid, M., Shakil Nobin, M. A., & Habib, M. A. (2019). Enhancing Quality of Service in SDN based on Multi-path Routing Optimization with DFS. In *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)* (pp. 1–5).

[Hu et al., 2014] Hu, Y., Wendong, W., Xiangyang, G., Liu, C. H., Que, X., & Cheng, S. (2014). Control traffic protection in software-defined networks. In *2014 IEEE Global Communications Conference* (pp. 1878–1883).

[Huang et al., 2016] Huang, H., Guo, S., Liang, W., Li, K., Ye, B., & Zhuang, W. (2016). Near-Optimal Routing Protection for In-Band Software-Defined Heterogeneous Networks. *IEEE Journal on Selected Areas in Communications*, 34(11), 2918–2934.

[Ibanez et al., 2018] Ibanez, G., Alvarez-Horcajo, A., Carral, J. A., Martinez-Yelmo, I., & Lopez-Pajares, D. (2018). Procedimiento de establecimiento y borrado de caminos múltiples disjuntos, de reenvío de tramas y puente de red. Universidad de Alcalá. Spanish patent ES2638292B2.

[Ibáñez & Rojas, 2013] Ibáñez, G. & Rojas, E. (2013). All-path bridging: Path exploration as an efficient alternative to path computation in bridging standards. In *2013 IEEE International Conference on Communications Workshops (ICC)* (pp. 1280–1285).

[IEEE, 1998] IEEE (1998). IEEE Standard for Local Area Network MAC (Media Access Control) Bridges. 802.1d - Spanning Tree Protocol.

[IEEE, 2001] IEEE (2001). IEEE Standard for Information Technology, Telecommunications and Information Exchange Between Systems, Local and Metropolitan Area Networks, Common Specifications, Part 3: Media Access Control (MAC) Bridges: Amendment 2, Rapid Reconfiguration. 802.1w - Rapid Spanning Tree Protocol.

[IEEE, 2012] IEEE (2012). IEEE Standard for Local and metropolitan area networks, Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks, Amendment 20: Shortest Path Bridging. 802.1aq - Shortest Path Bridging.

[IEEE, 2017] IEEE (2017). 802c-2017 - IEEE Standard for Local and Metropolitan Area Networks:Overview and Architecture–Amendment 2: Local Medium Access Control (MAC) Address Usage. 802c-2017.

[IEEE, 2018] IEEE (2018). IEEE Standard for Ethernet. 802.3-2018.

[International Organization for Standardization, 1991] International Organization for Standardization (1991). Information technology-Open systems interconnection-Local area networks- Medium Access Control (MAC) service definition. ISO/IEC 10039:1991.

[Itai & Rodeh, 1988] Itai, A. & Rodeh, M. (1988). The Multi-Tree Approach To Reliability In Distributed Networks. *Information and Computation*, 79(1), 43–59.

[ITU-T, 1994] ITU-T (1994). Information technology–Open Systems Interconnection–Basic Reference Model: The basic model. Recommendation X.200.

[Jiawei et al., 2018] Jiawei, W., Xiuquan, Q., & Guoshun, N. (2018). Dynamic and adaptive multi-path routing algorithm based on software-defined network. *International Journal of Distributed Sensor Networks*, 14(10), 1–10.

[Karaata, 2019] Karaata, M. H. (2019). An algorithm for finding two node-disjoint paths in arbitrary graphs. *Journal of Computing and Information Technology*, 27(3), 1–14.

[Kaur et al., 2014] Kaur, K., Singh, J., & Ghumman, N. S. (2014). Mininet as software defined networking testing platform. In *International Conference on Communication, Computing & Systems (ICCCS)* (pp. 139–142).

[Kawarabayashi et al., 2012] Kawarabayashi, K.-i., Kobayashi, Y., & Reed, B. (2012). The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2), 424–435.

[Krishnan & Silvester, 1993] Krishnan, R. & Silvester, J. A. (1993). Choice of allocation granularity in multipath source routing schemes. In *IEEE INFOCOM'93 The Conference on Computer Communications, Proceedings* (pp. 322–329).

[Lanning et al., 2014] Lanning, D. R., Harrell, G. K., & Wang, J. (2014). Dijkstra's algorithm and Google maps. In *Proceedings of the 2014 ACM Southeast Regional Conference* (pp. 1–3).

[Lemeshko et al., 2019] Lemeshko, O., Yeremenko, O., Yevdokymenko, M., & Sleiman, B. (2019). Enhanced Solution of the Disjoint Paths Set Calculation for Secure QoS Routing. In *2019 IEEE International Conference on Advanced Trends in Information Theory, ATIT 2019* (pp. 210–213).

[Liao & Tsai, 2018] Liao, Y. Z. & Tsai, S. C. (2018). Fast Failover with Hierarchical Disjoint Paths in SDN. In *2018 IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings* (pp. 1–7).: IEEE.

[Linux Foundation, 2020] Linux Foundation (2020). Open vSwitch in-band control. `https://docs.openvswitch.org/en/latest/topics/design/#in-band-control`. Accessed: August 2020.

[Lopez-Pajares et al., 2018] Lopez-Pajares, D., Alvarez-Horcajo, J., Rojas, E., Asadujjaman, A., & Martinez-Yelmo, I. (2018). Repositorio GitHub de Amaru. `https://github.com/gistnetserv-uah/Amaru/`. Uploaded: December 2018.

[Lopez-Pajares et al., 2019] Lopez-Pajares, D., Alvarez-Horcajo, J., Rojas, E., Asadujjaman, A., & Martinez-Yelmo, I. (2019). Amaru: Plug&play resilient in-band control for SDN. *IEEE Access*, 7, 123202–123218.

[Lopez-Pajares et al., 2018] Lopez-Pajares, D., Alvarez-Horcajo, J., Rojas, E., Carral, J. A., & Ibanez, G. (2018). Iterative Discovery of Multiple Disjoint Paths in Switched Networks with Multicast Frames. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)* (pp. 409–412).

[Lopez-Pajares et al., 2020] Lopez-Pajares, D., Alvarez-Horcajo, J., Rojas, E., Carral, J. A., & Martinez-Yelmo, I. (2020). One-shot multiple disjoint path discovery protocol (1S-MDP). *IEEE Communications Letters*, 24(8), 1660–1663.

[Manohar et al., 2002] Manohar, P., Manjunath, D., & Shevgaonkar, R. K. (2002). Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Communications Letters*, 6(5), 211–213.

[Marina & Das, 2006] Marina, M. K. & Das, S. R. (2006). Ad hoc on-demand multipath distance vector routing. *Wireless communications and mobile computing*, 6(7), 969–988.

[Martín et al., 2020] Martín, B., Sánchez, Á., Beltran-Royo, C., & Duarte, A. (2020). Solving the edge-disjoint paths problem using a two-stage method. *International Transactions in Operational Research*, 27(1), 435–457.

[Marukawa et al., 2011] Marukawa, J., Nomura, Y., Yamada, S., Terasawa, M., Okamoto, S., & Yamanaka, N. (2011). Scalable multi-path discovery technique for parallel data transmission in next generation wide area layer-2 network. In *2011 1st International Symposium on Access Spaces (ISAS)* (pp. 208–212).

[McKeown et al., 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2), 69–74.

[Medina et al., 2001] Medina, A., Lakhina, A., Matta, I., & Byers, J. (2001). BRITE: an approach to universal topology generation. In *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (pp. 346–353).

[Middendorf & Pfeiffer, 1993] Middendorf, M. & Pfeiffer, F. (1993). On the complexity of the disjoint paths problem. *Combinatorica*, 13(1), 97–107.

[Mills, 1983] Mills, D. (1983). DCN local-network protocols. RFC-891.

[Minkenberg & Gusat, 2012] Minkenberg, C. & Gusat, M. R. (2012). Multipath discovery in switched ethernet networks.

[Mohan et al., 2018] Mohan, P. M., Truong-Huu, T., & Gurusamy, M. (2018). Towards resilient in-band control path routing with malicious switch detection in SDN. In *2018 10th International Conference on Communication Systems and Networks, COMSNETS 2018* (pp. 9–16).

[Montero et al., 2017] Montero, R. S., Rojas, E., Carrillo, A. A., & Llorente, I. M. (2017). Extending the Cloud to the Network Edge. *Computer*, 50(4), 91–95.

[Moy, 1998] Moy, J. (1998). OSPF Version 2. RFC 2328.

[Mysore et al., 2009] Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., & Vahdat, A. (2009). PortLand: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4), 39–50.

[Narten et al., 2007] Narten, T., Nordmark, E., Simpson, W., et al. (2007). Neighbor discovery for IP version 6 (IPv6). RFC 4861.

[Naseri et al., 2019] Naseri, H., Sadoon, A., & Alireza, A. (2019). BSFS : A Bidirectional Search Algorithm for Flow Scheduling in Cloud Data Centers. *Information Systems & Telecommunication*, 7(3), 175–183.

[Nasipuri et al., 2001] Nasipuri, A., Castañeda, R., & Das, S. R. (2001). Performance of multipath routing for on-demand protocols in mobile ad hoc networks. *Mobile Networks and applications*, 6(4), 339–349.

[Nguyen et al., 2017] Nguyen, H., Phung, C., Secci, S., Felix, B., & Nogueira, M. (2017). Can MPTCP secure Internet communications from man-in-the-middle attacks? In *2017 13th International Conference on Network and Service Management, CNSM 2017* (pp. 1–7).

[Niven-Jenkins et al., 2009] Niven-Jenkins, B., Brungard, D., Betts, M., Sprecher, N., & Ueno, S. (2009). Requirements of an MPLS Transport Profile. *RFC 5654*, (pp. 1–31).

[Ochoa-Aday et al., 2019] Ochoa-Aday, L., Cervelló-Pastor, C., & Fernández-Fernández, A. (2019). eTDP: Enhanced Topology Discovery Protocol for Software-Defined Networks. *IEEE Access*, 7, 23471–23487.

[Ogier & Shacham, 1989] Ogier, R. & Shacham, N. (1989). A distributed algorithm for finding shortest pairs of disjoint paths. In *IEEE INFOCOM '89, Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies* (pp. 173–182).

[Oliva et al., 2018] Oliva, G., Cioaba, S., & Hadjicostis, C. N. (2018). Distributed Calculation of Edge-Disjoint Spanning Trees for Robustifying Distributed Algorithms Against Man-in-the-Middle Attacks. *IEEE Transactions on Control of Network Systems*, 5(4), 1646–1656.

[Omizo et al., 2016] Omizo, T., Watanabe, T., Akiyama, T., & Iida, K. (2016). ResilientFlow: Deployments of distributed Control Channel Maintenance Modules to recover SDN from unexpected failures. *IEICE Transactions on Communications*, 99(5), 1041–1053.

[Open Networking Foundation, 2020] Open Networking Foundation (2020). ONF. `https://opennetworking.org`. Accessed: January 2021.

[Open Networking Fundation, 2012] Open Networking Fundation (2012). Software-defined networking: The new norm for networks. *ONF White Paper*, (pp.1̃2).

[Orda & Sprintson, 2004] Orda, A. & Sprintson, A. (2004). Efficient Algorithms for Computing Disjoint QoS Paths. *IEEE International Conference on Computer Communications (INFOCOM)*, 1, 727–738.

[P4 Language Consortium and others, 2017] P4 Language Consortium and others (2017). P4 Runtime.

[Pakzad et al., 2016] Pakzad, F., Portmann, M., Tan, W. L., & Indulska, J. (2016). Efficient topology discovery in OpenFlow-based software defined networks. *Computer Communications*, 77, 52–61.

[Park et al., 2017] Park, Y., Nguyen, D. T., Kang, B., Lee, K., Lee, J., & Choo, H. (2017). A fast recovery scheme based on detour planning for in-band OpenFlow networks. *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017*, (pp. 1–5).

[Perlman & Eastlake, 2011] Perlman, R. & Eastlake, D. (2011). Introduction to TRILL. *The Internet Protocol Journal*, 14(3), 2–19.

[Pfaff et al., 2015] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., & Casado, M. (2015). The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (pp. 117–130).

[Plummer, 1982] Plummer, D. C. (1982). An ethernet Address Resolution Protocol. RFC 826.

[Poularakis et al., 2017] Poularakis, K., Iosifidis, G., Smaragdakis, G., & Tassiulas, L. (2017). One step at a time: Optimizing SDN upgrades in ISP networks. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (pp. 1–9).

[Prakash et al., 2019] Prakash, A., Kannan, R., & BAFNA, G. (2019). Disjoint path computation systems and methods in optical networks. Ciena Corp. US patent US10003867B2.

[Raiciu et al., 2011] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., & Handley, M. (2011). Improving datacenter performance and robustness with multipath TCP. *ACM SIGCOMM Computer Communication Review*, 41(4), 266–277.

[Ramdhani et al., 2016] Ramdhani, M. F., Hertiana, S. N., & Dirgantara, B. (2016). Multipath routing with load balancing and admission control in Software-Defined Networking (SDN). In *2016 4th International Conference on Information and Communication Technology (ICoICT)* (pp. 1–6).

[Raza & Lee, 2019] Raza, A. & Lee, S. (2019). Gate Switch Selection for In-Band Controlling in Software Defined Networking. *IEEE Access*, 7, 5671–5681.

[Rekhter, 1988] Rekhter, J. (1988). NSFNET backbone SPF based Interior Gateway Protocol. RFC 1074.

[Riley & Henderson, 2010] Riley, G. F. & Henderson, T. R. (2010). The *ns-3* network simulator. In K. Wehrle, M. Günes, & J. Gross (Eds.), *Modeling and Tools for Network Simulation* (pp. 15–34). Springer.

[Roberts & Kroese, 2007] Roberts, B. & Kroese, D. P. (2007). Estimating the Number of s-t Paths in a Graph. *Journal of Graph Algorithms and Applications*, 11(1), 195–214.

[Robertson & Seymour, 1995] Robertson, N. & Seymour, P. (1995). Graph Minors .XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, 63(1), 65 – 110.

[Rogers Communications Inc., 2018] Rogers Communications Inc. (2018). Rogers Fibre Backbone. `https://www.rogers.com/enterprise/wholesale#network`. Accessed: December 2018.

[Rojas et al., 2017] Rojas, E., Alvarez-Horcajo, J., Martinez-Yelmo, I., Arco, J. M., & Carral, J. A. (2017). GA3: scalable, distributed address assignment for dynamic data center networks. *Annals of Telecommunications*, 72(11), 693–702.

[Rojas et al., 2015] Rojas, E., Ibañez, G., Gimenez-Guzman, J. M., Carral, J. A., Garcia-Martinez, A., Martinez-Yelmo, I., & Arco, J. M. (2015). All-Path bridging: Path exploration protocols for data center and campus networks. *Computer Networks*, 79, 120 –132.

[Rojas et al., 2011] Rojas, E., Naous, J., Ibañez, G., Rivera, D., Carral, J. A., & Arco, J. M. (2011). Implementing ARP-path low latency bridges in NetFPGA. *ACM SIGCOMM Computer Communication Review*, 41(4), 444–445.

[Sakic et al., 2020] Sakic, E., Avdic, M., Van Bemten, A., & Kellerer, W. (2020). Automated bootstrapping of a fault-resilient in-band control plane. In *SOSR 2020 - Proceedings of the 2020 Symposium on SDN Research* (pp. 1–13).: Association for Computing Machinery, Inc.

[Sakic et al., 2018] Sakic, E., Đerić, N., & Kellerer, W. (2018). MORPH: An Adaptive Framework for Efficient and Byzantine Fault-Tolerant SDN Control Plane. *IEEE Journal on Selected Areas in Communications*, 36(10), 2158–2174.

[Savas et al., 2018] Savas, S. S., Tornatore, M., Dikbiyik, F., Yayimli, A., Martel, C. U., & Mukherjee, B. (2018). RASCAR: Recovery-Aware Switch-Controller Assignment and Routing in SDN. *IEEE Transactions on Network and Service Management*, (pp. 1222–1234).

[Schiff et al., 2015] Schiff, L., Schmid, S., & Canini, M. (2015). Medieval: Towards A Self-Stabilizing, Plug & Play, In-Band SDN Control Network. In *ACM Sigcomm Symposium on SDN Research (SOSR)*.

[Schiff et al., 2016] Schiff, L., Schmid, S., & Canini, M. (2016). Ground Control to Major Faults: Towards a Fault Tolerant and Adaptive SDN Control Network. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)* (pp. 90–96).

[Shaikh, 1995] Shaikh, S. (1995). Span-disjoint paths for physical diversity in networks. In *Proceedings IEEE Symposium on Computers and Communications* (pp. 127–133).

[Sidhu et al., 1991] Sidhu, D., Nair, R., & Abdallah, S. (1991). Finding disjoint paths in networks. *Conference on Communications architecture & protocols (SIGCOMM '91)*, (pp. 43–51).

[Smith & Thurlow, 2013] Smith, B. R. & Thurlow, L. (2013). Practical multipath load balancing with QoS. *2013 International Conference on Computing, Networking and Communications, ICNC 2013*, (pp. 937–943).

[Soliman et al., 2014] Soliman, M., Nandy, B., Lambadaris, I., & Ashwood-Smith, P. (2014). Exploring source routed forwarding in SDN-based WANs. In *2014 IEEE International Conference on Communications (ICC)* (pp. 3070–3075).

[Srivastava et al., 2004] Srivastava, S., Krithikaivasan, B., Beard, C., Medhi, D., Van De Liefvoort, A., Alanqar, W., & Nagarajan, A. (2004). Benefits of traffic engineering using QoS routing schemes and network controls. *Computer Communications*, 27(5), 387–399.

[Starobinski et al., 2003] Starobinski, D., Karpovsky, M., & Zakrevski, L. A. (2003). Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Transactions on Networking*, 11(3), 411–421.

[Steele & Allen, 2004] Steele, P. R. & Allen, C. J. (2004). *Handbook of Inca mythology*. ABC-CLIO.

[Su et al., 2017] Su, Y.-L., Wang, I.-C., Hsu, Y.-T., & Wen, C. H.-P. (2017). FASIC: A Fast-recovery, Adaptively Spanning In-band Control Plane in Software-Defined Network. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, (pp. 1–6).

[Suurballe, 1974] Suurballe, J. W. (1974). Disjoint paths in a network. *Networks*, 4(2), 125–145.

[Suurballe & Tarjan, 1984] Suurballe, J. W. & Tarjan, R. E. (1984). A Quick Method for Finding Shortest Pairs of Disjoint Paths. *Networks*, 14(2), 325–336.

[Taft-Plotkin et al., 1999] Taft-Plotkin, N., Bellur, B., & Ogier, R. (1999). Quality-of-service routing using maximally disjoint paths. *IEEE Seventh International Workshop on Quality of Service. IWQoS'99*, (pp. 119–128).

[Tanaka, 2014] Tanaka, J. (2014). Path Generating method, relay device, and computer product. Fujitsu Ltd. US patent US8665754B2.

[Tapolcai et al., 2019] Tapolcai, J., Retvari, G., Babarczi, P., & Berczi-Kovacs, E. R. (2019). Scalable and Efficient Multipath Routing via Redundant Trees. *IEEE Journal on Selected Areas in Communications*, 37(5), 982–996.

[Tapolcai et al., 2015] Tapolcai, J., Retvari, G., Babarczi, P., Berczi-Kovacs, E. R., Kristof, P., & Enyedi, G. (2015). Scalable and Efficient Multipath Routing: Complexity and Algorithms. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)* (pp. 376–385).

[Tilmans & Vissicchio, 2014] Tilmans, O. & Vissicchio, S. (2014). IGP-as-a-Backup for Robust SDN Networks. In *10th International Conference on Network and Service Management (CNSM) and Workshop* (pp. 127–135).

[Tolba, 2017] Tolba, A. (2017). Organizing Multipath Routing in Cloud Computing Environments. *International Journal of Advanced Computer Science and Applications*, 8(1), 455–462.

[Vahdat et al., 2015] Vahdat, A., Clark, D., & Rexford, J. L. (2015). A purpose-built global network: Google's move to sdn. *Queue*, 13(8), 73–98.

[Van Bemten et al., 2019] Van Bemten, A., Đerić, N., Varasteh, A., Blenk, A., Schmid, S., & Kellerer, W. (2019). Empirical predictability study of SDN switches. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (pp. 1–13).

[van der Kluit et al., 2017] van der Kluit, B. J., Holtzer, A. C. G., Gijsen, B. M. M., & Meeuwissen, H. B. (2017). Search for disjoint paths thorugh a network. K.MIZRA LLC. US patent US20170295088A1.

[Varga, 2010] Varga, A. (2010). OMNeT++. In K. Wehrle, M. Günes, & J. Gross (Eds.), *Modeling and Tools for Network Simulation* (pp. 35–59). Springer.

[Wang et al., 2011] Wang, H., Yu, Y., & Yuan, Q. (2011). Application of Dijkstra algorithm in robot path-planning. In *2011 Second International Conference on Mechanic Automation and Control Engineering* (pp. 1067–1069).

[Wang, 2004] Wang, X. (2004). Dijkstra Shortest Path Routing Library. in *MATLAB Central File Exchange*. `https://www.mathworks.com/matlabcentral/fileexchange/5550-dijkstra-shortest-path-routing`. Accesed: July 2020.

[Wang & Crowcroft, 1996] Wang, Z. & Crowcroft, J. (1996). Quality-of-service routing for supporting multimedia applications. *IEEE Journal on selected areas in communications*, 14(7), 1228–1234.

[Waxman, 1988] Waxman, B. M. (1988). Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9), 1617–1622.

[Xingyu et al., 2018] Xingyu, C., Huibing, Y., Peng, Y., & Shaoyong, G. (2018). An OSNR-based Master-slave Disjoint Routing Optimization Algorithm. In *2018 IEEE*

International Conference on Electronics and Communication Engineering, ICECE 2018 (pp. 71–75).

[Xu et al., 2018] Xu, H., Huang, H., Chen, S., Zhao, G., & Huang, L. (2018). Achieving high scalability through hybrid switching in software-defined networking. *IEEE/ACM Transactions on Networking*, 26(1), 618–632.

[Yan et al., 2015] Yan, J., Zhang, H., Shuai, Q., Liu, B., & Guo, X. (2015). HiQoS: An SDN-based multipath QoS solution. *China Communications*, 12(5), 123–133.

[Yan et al., 2017] Yan, Y., Bi, J., Zhou, Y., & Zhang, C. (2017). Gather: A Way to Optimize the Routing Process of In-band Control Network. In *Proceedings of the SIGCOMM Posters and Demos* (pp. 12–14).

[Yao et al., 2014] Yao, F., Wu, J., Venkataramani, G., & Subramaniam, S. (2014). A comparative analysis of data center network architectures. In *2014 IEEE International Conference on Communications (ICC)* (pp. 3106–3111).

[Zuo et al., 2019] Zuo, L., Zhu, M., Wu, C., Hou, A., & Cao, L. (2019). Bandwidth reservation for data transfers through multiple disjoint paths of dynamic HPNs. *Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019*, (pp. 2455–2460).