Universidad de Alcalá

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Contribución al diseño y evaluación de conmutadores avanzados basados en protocolos de exploración de caminos

Tesis Doctoral presentada por

Joaquín Álvarez Horcajo

2020

Universidad de Alcalá

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Contribución al diseño y evaluación de conmutadores avanzados basados en protocolos de exploración de caminos

Tesis Doctoral presentada por

Joaquín Álvarez Horcajo

Director

Isaías Martínez Yelmo

Alcalá de Henares, 2020

Me gustaría dedicar esta tesis a mi familia, porque sin ellos no lo hubiera conseguido. Su apoyo, su paciencia y sus ánimos son los pilares en los que siempre me apoyo. Del mismo modo, me gustaría hacer una mención especial para mi mujer, María, ya que sin su apoyo, amor y ánimos la realización de esta tesis no hubiera sido posible, y a mi hijo, Aitor, porque su alegría daba luz a cualquier día gris. Ellos son los que más fuerzas me han dado, día tras día, en los buenos momentos, pero especialmente en los malos. ¡Esta tesis va por vosotros dos!

"No todo es blanco o negro: Es gris. Todo depende del matiz, busca y aprende a distinguir"

Mägo de oz

Este trabajo ha sido parcialmente financiado por la Universidad de Alcalá a través del programa de Formación de Profesorado Universitario (FPU-UAH) y 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

It ain't no question of how just a matter of when

Stand Out - Tevin Campbell

Esta tesis es el fruto del esfuerzo y dedicación durante muchos meses, por ello quisiera comenzar agradeciendo a mi director, Isaías Martínez Yelmo, y a mi tutor, Juan Antonio Carral Pelayo, su paciencia, su sabiduría, sus ánimos y su apoyo, porque sin ellos no hubiera sido posible.

Del mismo modo me gustaría agradecer a todo el equipo de NETSERV la posibilidad de trabajar con ellos durante estos años. Gracias por compartir vuestros conocimientos conmigo. También quiero agradecer a mis compañeros del fondo de pasillo LE-34, quienes han sufrido las consecuencias de la tesis día tras día.

No quiero terminar esta sección, sin agradecer a mi familia todo el apoyo, cariño, ánimos y aliento. Sois el pilar donde me apoyo, el edificio donde me cobijo y el fuego que me da calor.

¡Muchas gracias a todos!

Índice general

Índice general **xi**

Índice de figuras **xvii**

Índice de tablas **xxi**

Índice de listados de código fuente **xxiii**

Índice de algoritmos **xxv**

Lista de acrónimos **xxix**

1 Introducción **1**

1.1 Motivación 1

1.2 Objetivos 3

1.3 Organización de la memoria 4

1.4 Contribuciones 5

2 Introduction **7**

2.1 Motivation 7

2.2 Objectives 9

2.3 Organization of the memory 10

2.4 Contributions 11

3 Estado del arte **13**

3.1 Redes de datos tradicionales 14

3.1.1 Introducción al modelo de red en capas tradicional o clásico 15

3.1.2 Modelos de referencia 16

3.1.2.1 Modelo OSI 16

3.1.2.2 Modelo TCP/IP 17
3.1.3 Capa de transporte 18
3.1.3.1 UDP: protocolo de datagramas de usuario 18
3.1.3.2 TCP: Protocolo de control de transmisión 19
3.1.3.3 Comparación UDP vs TCP 28
3.2 SDN: Redes definidas por software 28
3.2.1 La arquitectura SDN 29
3.2.2 Protocolo OpenFlow 30
3.2.2.1 OpenFlow Pipeline 30
3.2.2.2 Tablas OpenFlow 31
3.2.2.3 Mensajes OpenFlow 33
3.2.3 Switch OpenFlow 35
3.2.3.1 Lógica funcional de un switch OpenFlow 36
3.2.4 Implementaciones de switches software compatibles con OpenFlow 1.3 37
3.2.4.1 Basic OpenFlow Software Switch (BOFUSS) 38
3.2.4.2 Open vSwitch 40
3.2.5 Controlador 40
3.2.6 Tipos de controlador 41
3.2.7 Limitaciones de las arquitecturas SDN actuales 41
3.3 Redes de datos con arquitecturas híbridas 43
3.4 Migración de redes tradicionales a redes SDN 43
3.4.1 Control en redes mixtas o redes híbridas 44
3.4.2 Redes formadas por switches híbridos 45
3.5 Protocolos de descubrimiento de topologías 46
3.5.1 LLDP: Link Layer Discovery Protocol 46
3.5.2 CDP: Cisco Discovery Protocol 47
3.5.3 OFDP: OpenFlow Discovery Protocol 48
3.5.3.1 OFDPv2: OpenFlow Discovery Protocol v2 48
3.5.4 sOFTDP: Secure and Efficient Topology Discovery Protocol for SDN 49
3.5.5 SD-TDP: Software-Defined - Topology Discovery Protocol 49
3.5.6 ETDP: Enhanced Topology Discovery Protocol 50
3.6 Protocolos de árbol de expansión 52

3.6.1 STP: Spanning Tree Protocol 52
3.6.2 RSTP: Rapid Spanning Tree Protocol 54
3.6.3 MSTP: Multi Spanning Tree Protocol 55
3.7 Protocolos de establecimiento de caminos en capa de enlace 56
3.7.1 SPB: Shortest Path Bridging 56
3.7.2 TRILL: Transparent Interconnection of Lots of Links 56
3.7.3 ECMP: Equal Cost MultiPath 57
3.7.4 ARP-Path protocol 58

4 Mecanismo híbrido de recuperación frente a fallos 61
4.1 Basic OpenFlow Software Switch (BOFUSS) 62
4.1.1 Estudio del rendimiento de los switches software OvS y BOFUSS . 62
4.2 Switch híbrido ARP-Path/OpenFlow Software Switch (AOSS) 68
4.3 Recuperación híbrida 70
4.3.1 Mecanismo de recuperación 70
4.3.2 Trama Path Recovery 72
4.3.3 Diseño de la función realizada por el controlador SDN 73
4.3.4 Implementación en el controlador RYU 74
4.3.4.1 Definición y creación de la trama Path Recovery 74
4.3.4.2 Implementación lógica de la aplicación basada en controlador RYU 75
4.3.5 Diseño del mecanismo de recuperación en el conmutador 77
4.4 Evaluación 78
4.5 Resultados 80
4.5.1 Validación del funcionamiento 80
4.5.2 Tiempo de recuperación de la red 80
4.5.3 Tiempo de recuperación en el controlador SDN 82
4.6 Conclusiones 83

5 Encaminamiento de flujos TCP mediante exploración de caminos 85
5.1 TCP-Path 86
5.1.1 Diseño Funcional TCP-Path 88
5.1.2 Mecanismo anti-rebote en el proceso de difusión 90
5.1.3 Implementación de TCP-Path en un Switch Software 92

5.1.3.1 Procesamiento general de tramas en TCP-Path 93
5.1.3.2 Tratamiento de tramas TCP: SYN/SYN+ACK 94
5.1.3.3 Tratamiento de las tramas especiales Path Request y Path Reply 97
5.1.3.4 Tratamiento de los mensajes TCP 99
5.1.4 Evaluación 100
5.1.5 Resultados 100
5.1.5.1 Rendimiento en términos de BW y FCT 101
5.1.5.2 Tamaño de Tablas 105
5.1.5.3 Difusión de tramas 106
5.2 TFE: TCP-Path para *Elefantes* 108
5.2.1 Diseño funcional de TFE 108
5.2.2 Implementación de TFE en un switch SDN 109
5.2.3 Evaluación 110
5.2.4 Rendimiento respecto a ancho de banda medio (BW) y Flow Completion Time (FCT) 111
5.2.5 Tamaño de las Tablas 117
5.2.6 Sobrecarga por difusión de tramas de control 117
5.3 Mecanismo de recuperación 120
5.4 Conclusiones 121

6 Descubrimiento de caminos y topologías en redes con dispositivos híbridos SDN 123
6.1 TEDP: Tree Exploration Discovery Protocol 124
6.1.1 Funcionamiento básico de TEDP 124
6.1.2 Diferencias de funcionamiento entre modo centralizado y modo híbrido 126
6.2 Paquete de descubrimiento Path Discovery 127
6.3 Diseño e implementación del protocolo TEDP 128
6.3.1 Diseño e implementación como aplicación SDN 128
6.3.2 Diseño e implementación como sistema híbrido SDN/distribuido . . 130
6.4 Evaluación 131
6.4.1 Entorno de pruebas 132
6.4.2 Prueba de concepto 133
6.4.3 Pruebas de rendimiento 133
6.5 Conclusiones 136

7 Conclusiones y trabajos futuros **137**

7.1 Conclusiones 137

7.1.1 Mecanismo de recuperación en entornos híbridos SDN/Distribuidos 138

7.1.2 Protocolos de descubrimiento de caminos TCP-Path y TFE 139

7.1.3 Servicio de descubrimiento de caminos y topologías sobre redes con dispositivos híbridos SDN 141

7.2 Trabajos futuros 142

8 Conclusions and future work **143**

8.1 Conclusions 143

8.1.1 Recovery mechanism in hybrid SDN/Distributed environments . . . 144

8.1.2 Path discovery protocols TCP-Path and TFE 144

8.1.3 Path and topology discovery service for hybrid networks 146

8.2 Future work 146

Bibliografía **149**

A Artículos y patente **159**

Índice de figuras

3.1 Modelo de Capas 15
3.2 Estructura de capas del Modelo OSI 17
3.3 Comparación del modelo TCP/IP y modelo OSI 18
3.4 Cabecera UDP 19
3.5 Máquina de estados de TCP 20
3.6 Máquina estados del algoritmo de control de congestión Reno 22
3.7 Representación de la evolución de la ventana de congestión de Reno 24
3.8 Comparación de la evolución de las ventanas de congestión de Reno y Vegas 25
3.9 Ejemplo de evolución de una ventana de congestión aplicando BIC 25
3.10 Ejemplo de evolución de una ventana de congestión aplicando CUBIC . . . 27
3.11 Evolución de la función cúbica empleada por la ventana de congestión aplicando CUBIC 28
3.12 Arquitectura SDN 29
3.13 OpenFlow pipeline 31
3.14 Intercambio mensajes asimétricos 34
3.15 Intercambios de mensajes simétricos 35
3.16 Funcionamiento del canal OpenFlow 36
3.17 Diagrama de flujo funcional switch OpenFlow 1.3 36
3.18 Procesado interno de una trama en un switch SDN 37
3.19 Arquitectura de BOFUSS 38
3.20 Estructura interna de OvS 40
3.21 Ejemplo de una red mixta 44
3.22 Ejemplo de funcionamiento del protocolo LLDP 46
3.23 Trama LLDP 47
3.24 TLV opcionales en la trama LLDP 47

3.25 Ejemplo de funcionamiento protocolo CDP 47
3.26 Ejemplo de funcionamiento protocolo OFDP 48
3.27 Máquina de estados de SD-TDP 50
3.28 Máquina de estados de las interfaces en ETDP 51
3.29 Máquina de estados de los puertos en STP 53
3.30 Flags de trama RSTP 54
3.31 Trama TRILL 57
3.32 Fases del protocolo ARP-Path 59

4.1 Uso del procesador por parte de OvS, BOFUSS Original y BOFUSS Mejorado 64
4.2 Topología Spine-Leaf 4x4x20 65
4.3 Distribución de tamaño de flujos 65
4.4 Distribución porcentual de número total de flujos vs número total de *bytes* por flujo 66
4.5 Comparación del rendimiento entre OvS, BOFUSS Original y BOFUSS Mejorado 67
4.6 Diagrama de flujo switch AOSS 69
4.7 Recuperación híbrida 71
4.8 Estructura y Contenido de la trama *Path Recovery* 73
4.9 Diagrama de flujo del controlador para la recuperación híbrida 73
4.10 Nuevo diagrama de flujo del switch híbrido para introducir el nuevo mecanismo de recuperación 77
4.11 Diagrama de flujo de la función de recuperación híbrida 78
4.12 Topología en anillo 79
4.13 Comparación entre los tiempos de recuperación obtenidos con distintos mecanismos 81
4.14 Comparación entre el procesamiento necesario en controlador obtenidos con distintos mecanismos 82

5.1 Fases del protocolo TCP-Path 86
5.2 Estructura y contenido de la trama *Path Request* 87
5.3 Estructura y contenido de la trama *Path Reply* 87
5.4 Diagrama de flujo TCP-Path 88
5.5 Camino generado sin mecanismo anti-rebote 90
5.6 Diagrama de flujo switch TAOSS 93

5.7 Comparación de rendimiento entre TCP-Path y ECMP 102
5.8 Mejora del rendimiento obtenido por TCP-Path respecto a Equal Cost Multi-Path (ECMP) representada en % 103
5.9 Validación del modelo emulado 104
5.10 Mejora porcentual del BW medio obtenido por TCP-Path respecto a ECMP en los entornos de simulación y emulación a 10 Mbps 105
5.11 Tamaño de tabla de encaminamiento de TCP-Path 106
5.12 Porcentaje de difusión medio por Leaf 108
5.13 Diagrama de flujo TFE 109
5.14 Topología tres niveles basada en Altoona 110
5.15 Comparativa de rendimiento en la topología Spine-Leaf 4x4 con enlaces a 1 Gbps 112
5.16 Porcentaje de Mejora de rendimiento obtenido en la topología Spine-Leaf 4x4 a 1 Gbps respecto a ECMP 113
5.17 Comparativa de rendimiento en la topología de tres niveles a 10 Mbps . . . 114
5.18 Porcentaje de Mejora de rendimiento obtenido en la topología de tres niveles a 10 Mbps respecto a ECMP 115
5.19 Comparativa de BW en la topología Spine-Leaf 4x4 entre simulador y emulador a 10 Mbps 116
5.20 Tamaño de las tablas de los switches en la topología Spine-Leaf 4x4 a 1 Gbps 117
5.21 Porcentaje de broadcast generado por los protocolos de exploración en la topología Spine-Leaf 4x4 a 1 Gbps 120
5.22 Proceso de recuperación 121

6.1 Proceso llevado a cabo por el protocolo TEDP 125
6.2 Árbol de encaminamiento descubierto 126
6.3 Comparación trama LLDP vs TEDP 128
6.4 Diagrama de flujo de la lógica TEDP sobre controlador SDN 129
6.5 Diagrama de flujo de la lógica TEDP sobre sistema híbrido 131
6.6 Topologías formadas por 10 nodos 132
6.7 Topología GEANTS 132
6.8 Grafos topológicos 133
6.9 Arboles topológicos 134

Índice de tablas

3.1 Comparación de las características principales de TCP y UDP 28
3.2 Definición de Tabla de Flujos (*Flow Table*) 31
3.3 Ejemplo Tabla de Flujos (*Flow Table*) 32
3.4 Definición Tabla de Grupos (*Group Table*) 32
3.5 Ejemplo Tabla de Grupos (*Group Table*) 33
3.6 Definición de Tabla de Medidas (*Meter Table*) 33
3.7 Tabla de controladores 42

4.1 Resultados de Ancho de banda máximo y tiempo de procesamiento en OvS, BOFUSS Original y BOFUSS Mejorado 63
4.2 Configuración para la comparación de rendimiento entre OvS, BOFUSS Original y BOFUSS Mejorado 66
4.3 Media de paquetes perdidos durante el proceso de recuperación 80

5.1 Estructura y contenido entrada en tabla TCP-Path 88
5.2 Configuración para evaluación de TCP-Path 100
5.3 Calculo teórico del ratio de *bytes* de difusión utilizado por TCP-Path respecto al número de *bytes* de tramas unicast (%) 107
5.4 Configuración para evaluación 111
5.5 Calculo teórico del ratio de *bytes* de difusión en relación con la cantidad de *bytes* de tráfico unicast utilizado por TFE en comparación con TCP-Path en topología de dos niveles (%) 119
5.6 Calculo teórico del ratio de *bytes* de difusión en relación con la cantidad de *bytes* de tráfico unicast utilizado por TFE en comparación con TCP-Path en topologías *Vl2* (%) 119
5.7 Calculo teórico del ratio de *bytes* de difusión en relación con la cantidad de *bytes* de tráfico unicast utilizado por TFE en comparación con TCP-Path en topologías PortLand (%) 119

6.1 Resultados teóricos de Link Layer Discovery Protocol (LLDP) and Tree Exploration Discovery Protocol (TEDP) 135

6.2 Comparación experimental entre LLDP y TEDP 136

Índice de listados de código fuente

4.1 Codificación de la librería Path Recovery en el controlador 75
5.1 Definición de la estructura de las tramas TCP-Path 95
5.2 Nueva función de parseo dentro de la lógica de NebBee 96
5.3 Nueva casuística CustomNetPdl 96

Índice de algoritmos

4.1 Gestión de la *tabla de vecinos* 75

4.2 Búsqueda del nodo frontera del flujo 76

4.3 Creación y envío de una trama de recuperación 76

5.1 Inserción del valor de la capa en la que se sitúa el switch en los paquetes *Hello* 91

5.2 Proceso de difusión con anti-rebote 92

5.3 Selector de la siguiente función que debe manejar la trama 94

5.4 Procesamiento general de las tramas en TCP-Path 94

5.5 Encapsulado de las tramas TCP-Path 97

5.6 Manejador de las tramas propias del protocolo TCP-Path en switches no fronteras 98

5.7 Switch frontera TCP-Path 98

5.8 Desencapsulado de las tramas en TCP-Path 99

5.9 Encaminamiento de los mensajes TCP 99

5.10 Procesamiento general de las tramas en TFE 109

Lista de acrónimos

ACK	Acknowledgement Packet.
AOSS	ARP-Path OpenFlow Software Switch.
API	Application Programming Interface.
ARP	Address Resolution Protocol.
BEBA	BEhavioural BAsed forwarding.
BIC	Binary Increase Congestion control.
BPDU	Bridge Protocol Data Units.
BPDUv2	Bridge Protocol Data Units Version 2.
BW	BandWidth.
CDN	Content Delivery Network.
CDP	Cisco Discovery Protocol.
CLI	Common Languaje Interface.
DpId	Datapath Id.
ECMP	Equal Cost Multi-Path.
ETDP	Enhanced Topology Discovery Protocol.
FCT	Flow Completion Time.
IEEE	Institute of Electrical and Electronics Engineers.
IETF	Internet Engineering Task Force.
IP	Internet Protocol.
IS-IS	Intermediate System-Intermediate System.
ISO	International Organization for Standardization.
LACP	Link Aggregation Control Protocol.

LAN	Local Area Network.
LLDP	Link Layer Discovery Protocol.
MAC	Media Access Control.
MPLS	Multiprotocol Label Switching.
MSS	Maximum Segment Size.
MSTP	Multi Spanning Tree Protocol.
NFV	Network Function Virtualization.
ns-3	Network Simulator 3.
ODL	OpenDayLight.
OFDP	OpenFlow Discovery Protocol.
OFDPv2	OpenFlow Discovery Protocol Version 2.
ONF	Open Networking Foundation.
ONOS	Open Network Operating System.
OSI	Open Systems Interconnections.
OSPF	Open Shortest Path First.
OvS	Open vSwitch.
P4	Programming Protocol-independent Packet Processors.
PAST	Past: Scalable Ethernet For Data Centers.
RBridge	Routing Bridge.
RSPAN	Remote Switch Port ANalyzer.
RSTP	Rapid Spanning Tree Procotol.
RTT	Round-Trip Time.
SD-TDP	Software-Defined Topology Discovery Protocol.
SDN	Software-Defined Networking.
Smax	Valor del Incremento Máximo en el Sistema.
Smin	Incremento Mínimo en el Sistema.
SMTP	Simple Mail Transfer Protocol.
SNA	Systems Network Architecture.
sOFTDP	Secure and Efficient Topology Discovery Protocol for SDN.
SPB	Shortest Path Bridging.

STP	Spanning Tree Protocol.
TAOSS	TCP-Path OpenFlow Software Switch.
TCP	Transmission Control Protocol.
TCP/IP	Transmission Control Protocol/Internet Protocol.
CDF	Cumulative Distribution Function.
TEDP	Tree Exploration Discovery Protocol.
TFE	TCP-Path for Elephants.
TLS	Transport Layer Security.
TMN	Telecommunications Management Network.
ToR	Top of Rack.
TRILL	Transparent Interconnection of Lots of Links.
UDP	User Datagram Protocol.
Vc	Variable de Control de la Ventana de Congestión.
Vcmax	Valor Máximo de la Variable de Control de la Ventana de Congestión.
Vcmin	Valor Mínimo de la Variable de Control de la Ventana de Congestión.
Vcper	Valor de la Variable de Control de la Ventana de Congestión Antes de una Pérdida.
VLAN	Virtual Local Area Network.
XML	Extensible Markup Language.

Capítulo 1

Introducción

En este capítulo se presenta la motivación de la tesis, sus objetivos y su estructura.

1.1 Motivación

Las redes de datos han sufrido una gran evolución a lo largo de las últimas tres décadas. Esta evolución ha traído consigo una serie de mejoras en muchos aspectos de las redes, ya no sólo en cuanto a servicios, rendimiento o costes, sino también en cuanto a su propio diseño, arquitectura o despliegue.

Desde hace más de 30 años *Ethernet* [1] ha sido la tecnología más usada para la implantación y despliegue de redes de tipo Local Area Network (LAN) debido, entre otras razones, a su sencillez, a su bajo coste de instalación y a su alta velocidad de transmisión de datos. Debido a estas razones, su uso se ha extendido a redes de mayor envergadura como pueden ser Redes Empresariales, Data Center Networks (DCN) o Redes Metropolitanas. Sin embargo, esta expansión suele llevar asociada problemas como pueden ser: bajadas de rendimiento, interrupción del servicio o sobrecarga en los recursos debido a un deficiente reparto del tráfico que recorre la red, o simplemente, a la elección del camino que debe seguir un flujo.

Para resolver algunos de los problemas citados en el párrafo anterior, la comunidad científica ha propuesto diferentes soluciones a lo largo de los años:

- Respecto al problema de la aparición de bucles en topologías con múltiples caminos entre pares de nodos la comunidad científica propuso, en los años 90, el Spanning Tree Protocol (STP) [2]. Este protocolo define un mecanismo que permite crear un árbol de expansión, que garantiza la eliminación de los bucles al bloquear todos los enlaces no pertenecientes al árbol, provocando el desuso de todos aquellos enlaces que no pertenecen a dicho árbol. Esto provoca un desaprovechamiento de la capacidad de la red, así como la sobrecarga de aquellos enlaces que formen parte de dicho árbol.

- Para seleccionar un camino entre todos los caminos de igual coste existentes entre un par de nodos, durante la década de los 2000, la comunidad científica definió el algoritmo Equal Cost Multi-Path (ECMP) [3]. Este algoritmo, establecido en la actualidad como estándar *de facto*, permite seleccionar un camino entre todos los caminos de igual coste que existen entre dos nodos de la red. Para poder realizar esta elección, ECMP hace uso de otros protocolos de descubrimiento topológico que le faciliten los caminos posibles ya calculados como, por ejemplo, Open Shortest Path First (OSPF) [4].

Actualmente, la evolución de las redes lejos de estancarse está incluso acelerándose, aumentando su rendimiento y capacidades, pero también su nivel de complejidad en la configuración y mantenimiento de las mismas. Esto ha llevado a proponer arquitecturas que faciliten a los administradores su operación y mantenimiento. Para mitigar el gran aumento en su complejidad han surgido diferentes propuestas que intentan hacer más sencilla su gestión, como por ejemplo la arquitectura de redes definidas por software (Software-Defined Networking (SDN)) [5].

La arquitectura SDN rompe con la estructura clásica de las redes tradicionales, debido a que establece una estructura dividida en tres capas. La primera capa o *Capa de Aplicación*, permite al gestor de la red desarrollar los servicios y las políticas que se desean establecer en la red. La segunda capa o *Capa de Control*, define mediante un *middleware* todas las funciones y servicios que los desarrolladores tienen a su disposición para el control y la gestión de la red. Cabe citar a los controladores más importantes actualmente: Open Network Operating System (ONOS) [6], OpenDayLight (ODL) [7] y RYU [8]. Los dos primeros son de alto rendimiento y usados en entornos de producción, mientras que el último define un controlador más básico, pero que permite un prototipado mucho más rápido y sencillo. Y, por último, la *Capa de Infraestructura*, que contiene el *hardware* de la red, definiendo *hardware* de red como switches o nodos de red multicapa capaces de establecer conexión con el controlador.

SDN necesita de nuevos elementos para la constitución de su novedosa propuesta arquitectónica. Su punto central, la *Capa de Control*, establece la mayor diferencia con las redes tradicionales, ya que concentra en un único punto lógico (que no físico) toda la inteligencia de la red, es decir, los protocolos distribuidos necesarios para el correcto funcionamiento de las redes tradicionales ya no son tan necesarios, dado que ahora existe una unidad lógica capaz de conocer en todo momento los distintos estados de la red. Esta unidad lógica utiliza diferentes protocolos de comunicación para contactar con los switches SDN, así como protocolos de descubrimiento de red adaptados a este tipo de sistemas. El protocolo de comunicación más utilizado para la comunicación entre la *Capa de Control* y los diferentes nodos de la red es *OpenFlow* [9], el cual define los diferentes paquetes para el intercambio de información entre la *Capa de Control* y los switches SDN, así como la definición de un *pipeline* para el tratamiento de los diferentes paquetes de datos que

recorren la red. Esta estructura viene definida por un conjunto de tablas (reglas, medidas y grupos).

Otro de los cambios más importantes dentro de la arquitectura SDN es la necesidad que tiene la *Capa de Control* de conocer la topología de la *Capa de Infraestructura*. Esta necesidad viene dada por el uso que realizan las diferentes aplicaciones y servicios de red de dicha información, como por ejemplo *el algoritmo de Dijkstra*, el cual necesita conocer la información topológica de la red antes de poder calcular los diferentes caminos entre un origen y un destino. Para este propósito, los controladores llevan integradas aplicaciones tales como OpenFlow Discovery Protocol (OFDP) [10], un servicio de descubrimiento topológico basado en el protocolo LLDP [**?**].

1.2 Objetivos

El objetivo general planteado para el desarrollo de esta tesis doctoral consiste en establecer mejoras para los protocolos basados en mecanismos de exploración y que permitan o den apoyo al encaminamiento de las distintas tramas que componen un flujo de forma correcta. Estas propuestas pretenden mejorar aspectos fundamentales de las redes actuales, tales como aumentar la eficiencia de transmisión de los diferentes flujos que utilizan la red, disminuir su tiempo de estancia en ella o mejorar la distribución de los flujos para obtener un mejor balanceo de la carga.

Para lograr el objetivo general marcado en el párrafo anterior, se ha optado por establecer tres grandes bloques. El primero, pretende establecer mecanismos que mejoren la capacidad que tiene un sistema para recuperarse ante posibles fallos. El segundo, pretende proponer procesos o mecanismos encargados de establecer un correcto encaminamiento de los flujos dentro de la red. El tercero, trata de estudiar la problemática que existe en el proceso de descubrimiento de las topologías. Por lo tanto, se puede establecer como objetivos concretos los indicados a continuación:

- Se fija como primer objetivo el estudio, desarrollo e implementación de un nuevo mecanismo de recuperación que haga uso tanto del conocimiento global que tiene la *Capa de Control* de la arquitectura SDN acerca de la red como de la capacidad de diversificar el procesamiento de las redes tradicionales gracias a sus mecanismos distribuidos.

- Como segundo objetivo se desea diseñar e implementar un nuevo mecanismo capaz de establecer rutas desde un origen a un destino para cada flujo de la red y, además se fija como requisito para cumplir este segundo objetivo, que dichas rutas optimicen la distribución de carga en dicha red. Es bien sabido la existencia de dos tendencias para la creación de este tipo de mecanismos: basados en computación o basados en procesos distribuidos. Los mecanismos basados en cómputo se basan

en el cálculo de los caminos de manera centralizada lo que necesita de un *hardware* de gran potencia computacional. Mientras que los mecanismos basados en procesos distribuidos pretenden distribuir ese 'cálculo' entre todos los elementos de la red, relajando las características *hardware* necesarias. Dentro de este segundo grupo existen múltiples propuestas que, mediante el intercambio de paquetes entre los nodos de la red, intentan solventar el problema del establecimiento de caminos, como es el caso de STP. Del mismo modo, esta tesis quiere destacar los mecanismos basados en exploración de la red. Estos mecanismos, como su propio nombre indica, exploran la red mediante el envío de un paquete por todos sus enlaces, lo que provoca que estos mecanismos sean sencillos y veloces. Estas características, tan deseables para cualquier protocolo, se desean tener también como características del nuevo mecanismo, por lo que se fija como requisito para dar como cumplido este segundo objetivo el uso de la exploración mediante difusión como base para el establecimiento de los caminos. Así que, como resumen, se puede decir que como segundo objetivo se pretende buscar soluciones que hagan uso de la exploración mediante difusión para seleccionar los caminos por los que se van a encaminar cada una de las tramas que componen los diferentes flujos.

- Por último, se define como tercer objetivo el estudio y desarrollo de nuevos mecanismos para el descubrimiento de topologías que permitan proporcionar, no solo información sobre el grafo topológico de la red, sino también información adicional, como, por ejemplo: retardos, ancho de banda, etc. Este objetivo intenta dar solución al problema que existe tanto en redes tradicionales como en redes definidas por software sobre el descubrimiento parcial o total de la topología de la red. En el caso de las redes tradicionales, este conocimiento es requerido por cada uno de los switches que componen la red para el establecimiento de los diferentes caminos, mientras que, en el caso de las redes definidas por software, es la *Capa de Control* la que necesita conocer la topología completa para, posteriormente, ejecutar diferentes acciones tales como el establecimiento de las reglas en cada uno de los switches SDN para la definición de rutas y establecimiento de caminos. Pero en ambos casos, un error en el descubrimiento de la red puede ocasionar grandes problemas, ya que la comunicación entre los equipos podría no llegar a establecerse.

1.3 Organización de la memoria

Para dar una mejor visión de esta tesis, en esta sección se presenta la estructura en la que se encuentra organizada, así como un pequeño resumen de cada uno de los bloques:

Los capítulos 1 y 2 describen la motivación por la que se ha realizado el presente trabajo, además de exponer los diferentes objetivos que se desean llevar a cabo a lo largo del desarrollo de la tesis.

El capítulo 3 describe la situación del estado del arte a fecha de finalización de los diferentes trabajos realizados en la presente tesis. Dicho estado del arte se ha dividido en tres grandes bloques. El primero, describe la situación de las redes tradicionales o distribuidas, así como los protocolos de encaminamiento de mayor relevancia. El segundo bloque, introduce al lector en una nueva arquitectura, las redes definidas por software, la cual supone un cambio de paradigma en el mundo de las redes, ya que define una nueva arquitectura basada en una lógica centralizada. Del mismo modo, también se introduce al lector en el concepto de las redes híbridas donde se intenta ver que ambas arquitecturas pueden funcionar de forma conjunta, aprovechando las virtudes de cada una de ellas. Por último, se presenta el estado actual de los mecanismos y protocolos que permiten el descubrimiento de topologías, independientemente de la arquitectura de red utilizada.

El capítulo 4 propone un nuevo mecanismo de recuperación para arquitecturas de red, basadas en redes definidas por software. Este mecanismo aprovecha la potencia de los controladores introducidos por este tipo de arquitecturas junto a la versatilidad de los sistemas distribuidos.

El capítulo 5 describe dos nuevos mecanismos para el encaminamiento de los flujos. Estos protocolos utilizan como medio de descubrimiento de caminos la exploración y están pensados especialmente para flujos Transmission Control Protocol (TCP). También se expone en este capítulo un sistema de recuperación ante fallos especial para estos protocolos.

El capítulo 6 propone un nuevo mecanismo de descubrimiento de topologías de red. Dicho protocolo hace uso de los mecanismos de exploración para llevar a cabo su objetivo.

Los capítulos 7 y 8 se reservan para las conclusiones y líneas futuras de investigación. En estos capítulos se revisa el cumplimiento de los objetivos anteriormente expuestos, así como una evaluación de cada uno de ellos.

1.4 Contribuciones

Esta tesis ha contribuido a la publicación de cuatro artículos en revistas indexadas en el Journal Citation Reports (JCR) (dos Q1, un Q2 y un Q3) y de un artículo en un congreso, el cual además fue galardonado con el premio *Best Paper Award*:

1. Álvarez-Horcajo, Joaquin; Martínez-Yelmo, Isaías; Rojas, Elisa, Carral; Juan Antonio y López-Pajares, Diego (2017). New cooperative mechanisms for software defined networks based on hybrid switches. Transactions on Emerging Telecommunications Technologies, 28(8), e3150. (JCR Q3) [11]

2. Álvarez-Horcajo, Joaquín; López-Pajares, Diego; Arco, José Manuel ; Carral, Juan Antonio y Martínez-Yelmo, Isaías (2017). TCP-Path: Improving load balance by

network exploration. IEEE 6th International Conference on Cloud Networking (CloudNet), 1-6. (conference with *Best Paper Award*) [12]

3. Rojas, Elisa; Álvarez-Horcajo, Joaquín; Martínez-Yelmo, Isaías; Carral, Juan Antonio y Arco, José Manuel (2018). TEDP: An enhanced topology discovery service for software-defined networking. IEEE Communications Letters, 22(8), 1540-1543. (JCR Q2) [13]

4. Álvarez-Horcajo, Joaquín; López-Pajares, Diego; Martínez-Yelmo, Isaías; Carral, Juan Antonio y Arco, José Manuel (2019). Improving multipath routing of TCP flows by network exploration. IEEE Access. (JCR Q1) [14]

5. E. L. Fernandes, E. Rojas, J. Álvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, The Road to BOFUSS: The basic OpenFlow userspace software switch, Journal of Network and Computer Applications, vol.165, p. 102685, 2020. (JCR Q1) [15]

Por otro lado, esta tesis también ha contribuido a la generación de una patente:

1. Guillermo Ibáñez, Joaquín Álvarez, José Manuel Arco y José Manuel Giménez. "Procedimiento cooperativo, entre puentes y controlador, de reparación de caminos en fallo y puente de red". Patente de invención. Nº de patente: ES20160000530. Fecha de Concesión: 17/04/2018. [16]

Capítulo 2

Introduction

In this chapter, the motivation of the thesis, its objectives and its structure are presented.

2.1 Motivation

Data networks have undergone a great evolution over the last three decades. This evolution has brought a set of improvements in many aspects of data networks. These aspects are not only in terms of services, performance or cost, but also in its design, architecture or deployment.

Last thirty years *Ethernet* [1] has been the most used technology to the implementation and deployment of Local Area Networks (LAN) due to, among other reasons, the simplicity it offers, its low installation costs and its high data transmission speeds. Due to these reasons, its use has been extended to larger networks such as Enterprise Networks, Data Center Networks (DCN) or Metropolitan Networks. However, this expansion usually comes with several associated problems, such as, a decrease in performance, service interruptions or an overload on the resources due to the load balance in the network is not done accurately.

To solve some problems cited in the previous paragraph, different solutions have been proposed by the scientific community over the years:

- Regarding to the problem of the appearance of loops in network topologies with multiple paths between a pair of switches, the scientific community proposed, in the 90s, the Spanning Tree Protocol (STP) [2]. This protocol defines a mechanism that creates an expansion tree that guarantees the removal of loops blocking all the links that don't belong to the tree. This fact produces a waste of the network capacity, besides the overuse of the links that are part of the tree.

- In relation to the problem with the path selection among multiple paths that exist in current networks between a pair of switches, in the 2000s, the scientific community

defined Equal Cost Multi-Path (ECMP) [3] algorithm. This algorithm, currently established as the *de facto* standard, allows choosing the equal-cost path between two switches of the network. ECMP uses other topology discovery or routing protocols to obtain the necessary topological information, e.i.,OSPF [4], to later apply a balanced election algorithm.

Nowadays, the network evolution is far from stalling, it is even accelerating and increasing the performance of networks and their capabilities, but also the complexity level of their configuration and management. These facts have led to propose new architectures that ease the operation and maintenance of the networks to the network managers. To address the increase of the complexity, the Software-Defined Networking (SDN) [5] architecture is being adopted and improved nowadays.

The Software Defined Networking (SDN) architecture breaks with the classical structure of the legacy network architectures since it defines a structure divided into three different layers. The first layer, named *Application Layer*, allows the network manager to implement new services and policies that are required by the network managers. The second layer, named *Control Layer*, defines all the functions and services that developers can use to control the network by a middleware. It is worth mentioning the most important controllers that we can find nowadays: ONOS [6], ODL [7] and RYU [8]. While, the first and second are high-performance controllers used in production environments. The last one (RYU) is an *OpenFlow* [9] basic controller which allows creating prototypes faster and easier than the others, but it offers worse performance. Finally, the *Infrastructure Layer* is composed of the network devices in the underlying network infrastructure.

The SDN architecture needs new elements to the support of its novel architectural proposal. Its main key, the *Control Layer*, establishes the most important difference between them and the legacy networks since it concentrates on a unique logical point (not physical) all the network intelligence, which in legacy networks is performed by distributed protocols. This *Control Layer* uses different communication protocols to communicate with the SDN switches, as well as network discovery protocols are adapted to this new paradigm. The most used communication protocol used in the *Control Layer* with all SDN switches is *OpenFlow* [9], which defines the different necessary packets to achieve the communication as well as the internal node structure. This structure is defined by the table sets (rules, meters, and groups), and the definition of a pipeline to process the incoming data packets.

Another of the most important changes inside the SDN architecture is the need to know the network topology by the *Control Layer*. This need is given by the use that different applications and network services make of the topology information, for instance, the *Dijkstra algorithm*. This algorithm needs to know the topological information before calculating the different paths between any source and any destination. To get the information about the network topology, the controllers have integrated applications such

as OFDP. This application is a topology discovery service based on the LLDP protocol [**?**] which gives to the *Control Layer* all the topology information.

2.2 Objectives

The general objective fixed to the development of this Ph.D. thesis consists of establishing improvements into the current protocols based on exploration mechanisms and allowing or supporting to forward the different frames that compose a flow. This thesis pretends to improve some basic aspects of the current networks, such as increasing the efficiency of transmission, decreasing the Flow Completion Time (FCT) of flows, or improving the network load balance.

To achieve the general objective fixed in the previous paragraph, it has been decided to define three different blocks. The first attempts to define a mechanism that improves the network capacity to recover the communication between two different end-devices in the presence of possible failures. The second one intends to propose processes or mechanisms which can establish paths to forward different flows. Finally, the last one tries to study the current topology discovery process problems. Hence, the objectives indicated in the following paragraphs can be set as specific objectives:

- It is set as the first objective, the study, development and implementation of a new recovery mechanism that uses both the knowledge about the network that the *Control Layer* of the SDN architectures has and the distributed processing of certain aspects derived from traditional legacy networks.

- As a second objective, it is desired to design and implement a new mechanism capable of establishing paths for network flows and optimising the network load balance to improve the overall network performance. It is well-known the existence of two tendencies for the proposed type of mechanism: computer-based or exploration-based solutions. Computer-based mechanisms are based on calculating the paths in a centralized way that requires high-powered hardware. The distributed mechanisms distribute this 'calculation' among all the elements of the network, relaxing the necessary hardware characteristics. Within this second group, there are multiple proposals that, by exchanging packets between network switches, try to solve the problem of establishing paths, such as STP. Likewise, it is worth highlighting the mechanisms based on path exploration. These mechanisms, as its name suggests, explore the network by sending a packet through all its links. This fact makes these mechanisms simple and fast. These characteristics, which are so desirable for any protocol, are requirements for the desired new mechanism. Hence, the use of network exploration based on flooding frames as the key to the establishment of paths is selected as a requirement to achieve the second objective. As a summary, it can be

said that the second objective intends to find solutions that use the exploration by frame diffusion to select the paths to stablish the flows in the network.

- Finally, the third objective is the study and, and the development of a new mechanism to discover the topology of the underlying network that allows giving not only the information about the network topology graph but also additional information, such as delays, bandwidth, etc. This objective tries to give a solution to the current problem that exists in both architectural networks (legacy and SDN) about the partial or full topology discovery. In the case of legacy networks, each switch that composes the network requires this knowledge to establish the different paths. Whereas, in the case of SDN networks, it is the *Control Layer* that needs to know the complete topology to carry out different actions, such as installing the right forwarding rules. But in both cases, an error in network discovery could cause major problems, since communications can be affected because, among other things, they may not be established.

2.3 Organization of the memory

To give a better vision of this thesis, this section presents the structure of this Ph. D. thesis, which has been divided into the following chapters as follows:

Chapters 1 and 2 describe the motivation of this Ph.D. thesis in addition to the different objectives that are intended to accomplish on at the end of its execution.

Chapter 3 describes the situation of the state-of-art at the end date of the distinct jobs made in this thesis. The state-of-art has been divided into three parts. The first one states the situation of legacy or distributed networks, as well as the most important forwarding protocols. The second one introduces the reader to a new architecture, the Software Defined Networking, which generates an inflection point in the network world since it defines new elements and new points of view. Likewise, it also leads to the reader in the hybrid network concept in which it is tried to show how both architectures can work together, taking advantage of the benefits of each one. Finally, the current status of mechanisms and protocols that allows the topology discovery independently of the network architecture used is presented.

In chapter 4 a new recovery mechanism for SDN network architectures is elucidated. This mechanism leverages the topology knowledge that any SDN *Control Layer* has along with the versatility of distributed systems.

Chapter 5 describes two new mechanisms to forwarding the different flows. The exploration is used by these protocols to create the paths. These protocols have thought especially to TCP flows. Also, a new special recovery system for these protocols when any fail appears is explained.

In chapter 6 a new network topology discovery mechanism is proposed. This protocol uses an exploration mechanism by broadcasting to achieve its objective.

Chapters 7 and 8 present the conclusions and future work derived from this Ph.D. thesis, compliance of all objectives.

2.4 Contributions

This thesis has contributed to the publication of four articles in journals indexed into the JCR ([14] and [15] have been published in a Q1 journal, [13] has been published in a Q2 journal and [11] has been published in a Q3 journal) and a congress paper [12] which obtained the *Best Paper Award* :

1. Álvarez-Horcajo, Joaquin; Martínez-Yelmo, Isaías; Rojas, Elisa, Carral; Juan Antonio y López-Pajares, Diego (2017). New cooperative mechanisms for software defined networks based on hybrid switches. Transactions on Emerging Telecommunications Technologies, 28(8), e3150. (JCR Q3) [11]

2. Álvarez-Horcajo, Joaquín; López-Pajares, Diego; Arco, José Manuel ; Carral, Juan Antonio y Martínez-Yelmo, Isaías (2017). TCP-Path: Improving load balance by network exploration. IEEE 6th International Conference on Cloud Networking (CloudNet), 1-6. (conference with *Best Paper Award*) [12]

3. Rojas, Elisa; Álvarez-Horcajo, Joaquín; Martínez-Yelmo, Isaías; Carral, Juan Antonio y Arco, José Manuel (2018). TEDP: An enhanced topology discovery service for software-defined networking. IEEE Communications Letters, 22(8), 1540-1543. (JCR Q2) [13]

4. Álvarez-Horcajo, Joaquín; López-Pajares, Diego; Martínez-Yelmo, Isaías; Carral, Juan Antonio y Arco, José Manuel (2019). Improving multipath routing of TCP flows by network exploration. IEEE Access. (JCR Q1) [14]

5. E. L. Fernandes, E. Rojas, J. Álvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, The Road to BOFUSS: The basic OpenFlow userspace software switch, Journal of Network and Computer Applications, vol.165, p. 102685, 2020. (JCR Q1) [15]

Likewise, this thesis has contributed to the definition of a patent:

1. Guillermo Ibáñez, Joaquín Álvarez, José Manuel Arco y José Manuel Giménez. "Procedimiento cooperativo, entre puentes y controlador, de reparación de caminos en fallo y puente de red". Patente de invención. ID Number: ES20160000530. Date: 17/04/2018. [16]

Capítulo 3

Estado del arte

En este capítulo se presenta y revisa el estado del arte de los conmutadores avanzados de red dentro del marco tecnológico de la tesis. La revisión del estado del arte se va a dividir en cuatro grandes bloques: el primero concentra el estudio acerca de las redes de datos distribuidas, el segundo se centra en el estudio de las redes definidas por software, el tercer bloque fija su interés en el estudio de los distintos mecanismos para el descubrimiento de topologías de red y el cuarto bloque realiza la revisión de los protocolos de generación de caminos a nivel de enlace.

El primer bloque introduce las redes de datos tradicionales. Para ello, comienza con un primer punto donde se presentan los *sistemas distribuidos*, continua con un segundo punto en el que se realiza una revisión de los distintos métodos que ha seguido la comunidad científica para el diseño de las redes. Este segundo punto muestra la evolución de las redes desde un diseño basado en *hardware*, donde el peso de la red recaía principalmente en el diseño de los propios equipos hasta un diseño basado en *capas*, donde ese peso se diluye entre todos los elementos que componen el modelo. El tercer y último punto presenta una revisión del estado del arte referente a los modelos actuales basado en capas, de los cuales se quiere destacar: el modelo Open Systems Interconnections (OSI), modelo establecido como estándar por la International Organization for Standardization (ISO), y el modelo Transmission Control Protocol/Internet Protocol (TCP/IP), modelo establecido como estándar de facto. Para finalizar este primer bloque, se presenta un revisión de los protocolos de transporte más relevantes dentro del modelo TCP/IP, haciendo especial hincapié en el protocolo TCP y sus diferentes algoritmos de control de congestión.

El segundo bloque ofrece un resumen del nuevo paradigma introducido gracias a la aparición de la arquitectura SDN. Primero se revisa el nuevo modelo arquitectónico definido dentro del paradigma SDN. Este nuevo modelo define una separación de los planos de control y de datos, creando para ello una nueva entidad, el controlador, el cual pretende, por un lado, centralizar el conocimiento de la red, mientras que, por otro lado, pretende facilitar también su gestión, operación y mantenimiento. Luego se revisa *OpenFlow*, el protocolo de comunicación más utilizado para el intercambio de información

entre el *Plano de Control* y los dispositivos de red. La exposición se finaliza con un pequeño repaso de los distintos controladores que existen en el mercado, así como de sus características más relevantes. Por último, el bloque termina con la presentación de un modelo de red híbrido donde cohabitan elementos pertenecientes a las redes tradicionales y elementos compatibles con la arquitectura SDN.

El tercer bloque presenta un resumen de los diferentes mecanismos encargados de descubrir y proporcionar información sobre topologías de red. Estos mecanismos son vitales tanto para las arquitecturas tradicionales como para las redes SDN, ya que en ambos planteamientos existen funcionalidades, tales como el encaminamiento de flujos, que necesitan conocer información acerca de la topología (enlaces, switches, saltos, etc..) para poder llevarse a cabo correcta y eficientemente.

Para finalizar, en el último bloque se estudian los mecanismos de generación de caminos en capa dos (capa de enlace en el modelo TCP/IP). Este conjunto de mecanismos tiene como objetivo principal establecer el camino entre un par de switches dentro de las redes. Este estudio se inicia con una revisión de los diferentes mecanismos basados en árboles de expansión y acaba con una revisión de los protocolos de establecimiento de caminos en capa de enlace propiamente dichos.

3.1 Redes de datos tradicionales

Los sistemas de la información han sufrido una gran transformación en las últimas décadas, pasando de grandes ordenadores hasta los sistemas informáticos actuales con un tamaño mucho más pequeño y con un rendimiento infinitamente superior. Del mismo modo, las necesidades de los usuarios actuales también han evolucionado, ahora el usuario no solo desea un sistema aislado donde operar, sino que busca un sistema capaz de comunicarse con otros elementos externos, tales como sensores o proveedores de servicios, desde cualquier punto del planeta y, además, que dicha conexión sea rápida, fiable y eficiente. Estos nuevos *requisitos* provocaron que la comunidad científica realizara un profundo estudio que dio como resultado el nacimiento de las redes conocidas como *redes de datos tradicionales*, definiendo para ello todas sus partes, así como las relaciones entre ellas.

En la literatura se pueden encontrar múltiples definiciones que intentan explicar el concepto de *modelo arquitectónico de un sistema distribuido* pero de todas ellas, en esta tesis se desea enfatizar la definición dada por *George Coulouris* [17] la cual establece que el modelo arquitectónico de un sistemas distribuido es '*la estructura de cada uno de los elementos específicos por separado*'. A esta definición, a ojos de esta tesis, se le debe cumplimentar con las características dadas por *De Ugarte* [18] donde establece que, además '*debe cumplir la función de simplificar y abstraer las funciones de dichos componentes dentro del sistema, considerando posteriormente la ubicación de ellos y sus interrelaciones. Por lo tanto, se puede decir que todos los componentes del sistema no*

dependen unos de otros, sino que todos dependen de todos sin perder la libertad de trabajar de forma unitaria'.

Los sistemas distribuidos presentan importantes propiedades [19], de las cuales se desea citar las tres más relevantes para esta tesis. La primera, la *concurrencia* establece que los recursos que existen dentro del sistema deben poder ser utilizados por todos los usuarios de forma simultánea; La segunda, la *carencia de reloj global* define la no existencia de un reloj global que defina una temporización en el sistema general, sino que dicha temporización es más bien distribuida; y por último, *fallos independientes de los componentes* que estipula que el fallo en un componente no debe impedir la ejecución de las acciones en el resto de elementos que componen el sistema distribuido.

3.1.1 Introducción al modelo de red en capas tradicional o clásico

La complejidad en la lógica de los equipos que conforman las redes ha aumentado con el paso de los años llegando a disponer de una infinidad de funcionalidades dentro de ellos. Por ello, el diseño inicial basado en hardware donde los equipos físicos tenía un mayor peso a la hora de establecer el diseño de las redes ya no es válido, y ha sido reemplazado por un modelo basado en capas. El concepto de capa ha sido tratado por múltiples autores, pero en lo que respecta a esta tesis cabe citar la definición propuesta por *Tanenbaum* [20], que define el concepto de *capa* como un conjunto de servicios ofrecidos a los distintos usuarios pero con la peculiaridad de que no se les proporciona información del estado, algoritmos e información de los procesos internos del sistema. En otras palabras, lo define como si se tratase de una caja negra donde el usuario conoce únicamente sus entradas y salidas.

Para una mejor comprensión se presenta la figura 3.1, en ella se observa: un modelo basado en capas básico, donde cada una de esas capas puede ser, a su vez, cada uno de los consumidores de los servicios de otra capa. Esto significa que la capa de nivel dos puede ofrecer servicios y funcionalidades a las capas adyacentes (niveles uno y tres) pero también puede ofrecer servicios a las capas establecidas al mismo nivel, pero situadas en equipos distintos y, por lo tanto, no es necesario que las capas estén directamente conectadas entre sí, sino que se puede establecer lo que se conoce como *comunicación virtual* (ya que no es directa) entre pares de capas homónimas.

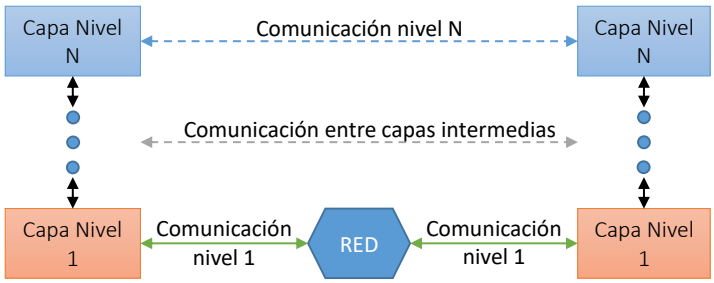


Figura 3.1: Modelo de Capas

3.1.2 Modelos de referencia

Este sistema de capas permite simplificar, dividir y ordenar las tareas a realizar por las redes, generando diferentes módulos según las necesidades, funcionalidades y características de las mismas. Estas razones han llevado a que su estructura modular se haya establecido como la base para los modelos de referencia [21] tales como, el Modelo OSI, el modelo TCP/IP, Systems Network Architecture (SNA) o Telecommunications Management Network (TMN), entre otros. En lo que respecta al estado del arte del marco tecnológico de esta tesis se ha tomado como eje central el estudio los Modelos OSI y TCP/IP.

3.1.2.1 Modelo OSI

El Modelo OSI [22], estandarizado por ISO, establece un sistema modular compuesto por siete capas: capa física, capa de enlace, capa de red, capa de transporte, capa de sesión, capa de presentación y capa de aplicación, así mismo también establece las funciones principales de cada una de las capas [21]:

- *Capa Física*: esta capa es la encargada de gestionar todas las funciones que tienen relación con las transmisiones de datos de una forma física.
- *Capa de Enlace*: esta capa dispone de todas las funciones necesarias para establecer la comunicación entre los diferentes elementos físicos, como, por ejemplo: direccionamiento físico de los datos, acceso al medio y detección de errores en la transmisión.
- *Capa de Red*: es la capa encargada de hacer llegar los datos a su destino, aunque no estén conectados de una manera directa. En esta capa encontramos las funciones de enrutamiento y control de congestión.
- *Capa de Transporte*: capa encargada de suministrar los mecanismos necesarios para *transportar* la información de extremo a extremo [23]. Esta capa también proporciona las funciones de control de flujo y control de errores, pero a diferencia de la anterior, se realiza extremo a extremo.
- *Capa de Sesión*: es la capa cuya función principal es la gestión, control y mantenimiento de la conexión (en el sentido lógico de la palabra) creada entre dos o más entidades.
- *Capa de Presentación*: esta capa permite abordar la representación de los datos solucionando así el problema de la heterogeneidad de las diferentes fuentes de información.
- *Capa de Aplicación*: esta capa engloba todos los servicios finales ofertados.

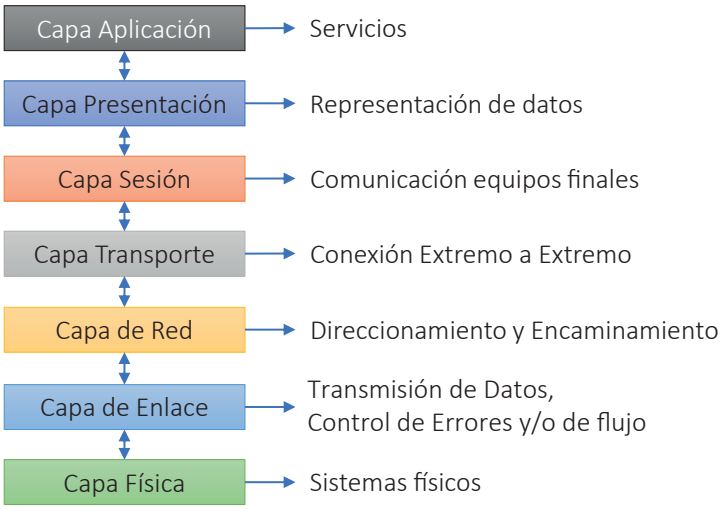


Figura 3.2: Estructura de capas del Modelo OSI

El modelo OSI también establece las relación y posición que ocupa cada una de las capas dentro de su modelo, para comprender mejor las relaciones que existen entre las distintas capas así como su posición dentro del modelo se presenta la figura 3.2.

3.1.2.2 Modelo TCP/IP

El modelo TCP/IP [24] es el modelo de referencia *de facto*. Este modelo recibe su nombre gracias a sus dos protocolos más importantes. Al igual que en el caso de modelo OSI, el modelo TCP/IP también se estructura en capas, pero a diferencia de OSI solo dispone de cuatro capas [20], esta diferencia se muestra mediante la comparación representada en la figura 3.3:

- *Capa de Acceso al Medio*: en esta capa el modelo TCP/IP unifica las funcionalidades de las capas física y de enlace del modelo OSI.
- *Capa de Red*: al igual que en el modelo OSI presentado en el apartado anterior, la capa de red se encarga principalmente del enrutamiento de los paquetes. El protocolo principal que rige esta capa es Internet Protocol (IP).
- *Capa de Transporte*: las funciones asignadas a esta capa son las mismas que las asignadas por el modelo OSI a su capa homónima. Sus protocolos más destacados son: TCP y User Datagram Protocol (UDP), los cuales serán explicados en las siguientes subsecciones.
- *Capa de Aplicación*: esta capa engloba a las tres últimas capas establecidas en el modelo OSI (Sesión, Presentación y Aplicación).

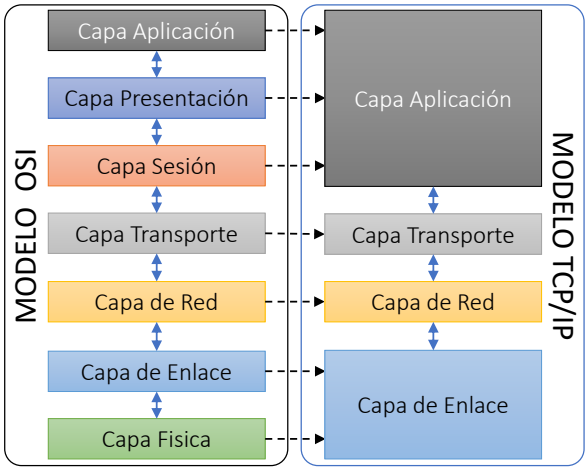


Figura 3.3: Comparación del modelo TCP/IP y modelo OSI

3.1.3 Capa de transporte

Una vez introducidos los modelos de referencia basados en capas, se va a centrar la atención en la capa de transporte del modelo TCP/IP, este hecho se debe a que dos de los protocolos propuestos en esta tesis se centran, principalmente, en el manejo de aquellos paquetes que disponen de una cabecera TCP. La capa de transporte es el primer nivel que considera comunicaciones extremo a extremo y, a diferencia de las capas inferiores, sitúa su complejidad en los equipos finales de las comunicaciones.

La capa de transporte del modelo TCP/IP, a diferencia de la capa de nivel de red donde solo existe un protocolo (IP), dispone de dos grandes protocolos de transporte diferentes: UDP, un protocolo sencillo que no ofrece ningún tipo de garantías ni control y, TCP, un protocolo mucho más robusto y pesado que el anterior, pero que ofrece servicios adicionales como pueden ser control de congestión y control de errores extremo a extremo.

3.1.3.1 UDP: protocolo de datagramas de usuario

El primer protocolo de transporte que se va a presentar es el protocolo UDP [25], un protocolo sencillo que no ofrece ningún tipo de garantía de entrega, ni de fiabilidad, pero, al contrario que otros protocolos de transporte, ofrece una transmisión con apenas sobrecarga y con un retardo mínimo. Esto hace que UDP sea especialmente útil para aplicaciones que no necesiten de un sistema que garantice fiabilidad o para aplicaciones que hagan uso de tramas *multicast o multidifusión*.

Las Service Data Units (SDUs) disponen de una cabecera formada por cuatro campos: puerto origen, puerto destino, longitud del paquete incluyendo cabeceras y *checksum*, como código de comprobación. Estos campos, así como un ejemplo del valor que puede tomar cada uno de ellos, se muestran en la figura 3.4.

Puerto Origen	Puerto Destino	Longitud del Mensaje	CheckSum
1 ->65535	1 ->65535	Longitud Datos + 8 Cabecera	Suma de Comprobación

Figura 3.4: Cabecera UDP

3.1.3.2 TCP: Protocolo de control de transmisión

TCP [26] es el protocolo de transporte más conocido y utilizado actualmente (más del 85 % de las transmisiones se realizan gracias a él), se trata de un protocolo orientado a conexión. Este protocolo permite transmisiones full-dúplex gracias al uso de la técnica *piggybacking* y, además, proporciona fiabilidad gracias al control de flujo y de errores, sin embargo, no proporciona otras características como puede ser seguridad, pero pueden introducirse mediante el uso de otros protocolos compatibles con él.

TCP tiene la capacidad [21] de controlar las transmisiones de datos entre equipos finales de forma muy minuciosa y exacta gracias a los diferentes mecanismos que integra, pero a cambio, dichos mecanismos introducen un mayor coste computacional, una mayor complejidad y menor velocidad en las transmisiones. Para llevar a cabo este control, TCP tiene implementado las siguientes funcionalidades [21]: mecanismos de apertura y cierre de conexiones para su administración y control, funciones para mitigar la congestión de red y, por último, funciones para controlar las posibles pérdidas de paquetes durante una transmisión. A continuación, se explica un poco más en profundidad cada una de esas funcionalidades.

Las funciones básicas que requiere TCP para poder afirmar que dispone de mecanismos capaces de *administrar y controlar las conexiones* parten de la definición que se le ha otorgado como protocolo *orientado a conexión*, es decir, parte de la base de la existencia de tres fases muy definidas dentro del protocolo. La primera, una fase mediante la cual se establezca la conexión entre equipos (*fase de establecimiento*), en esta fase los distintos dispositivos implicados en la comunicación negocian alguno de los parámetros de la comunicación tales como, el tamaño de mensaje o de las ventanas de transmisión. La segunda fase, requiere de un proceso de transmisión de datos propiamente dicho. Por último, la tercera fase necesita de un mecanismo de cierre de conexión (*fase de cierre de conexión*).

El proceso llevado a cabo por TCP para gestionar las conexión se presenta en la figura 3.5 [20] donde queda reflejado el diagrama de flujo que sigue TCP para establecer, comunicar y cerrar de forma correcta cualquier conexión.

Dentro de este diagrama (figura 3.5 [20]) se observan dos caminos diferentes, el primero representado por las líneas discontinuas definen el diagrama de estados seguido por los sistemas que actúan como receptores de la conexión; mientras que el diagrama ilustrado por las líneas continuas representa los estados definidos para los sistemas que actúan como emisores de la conexión.

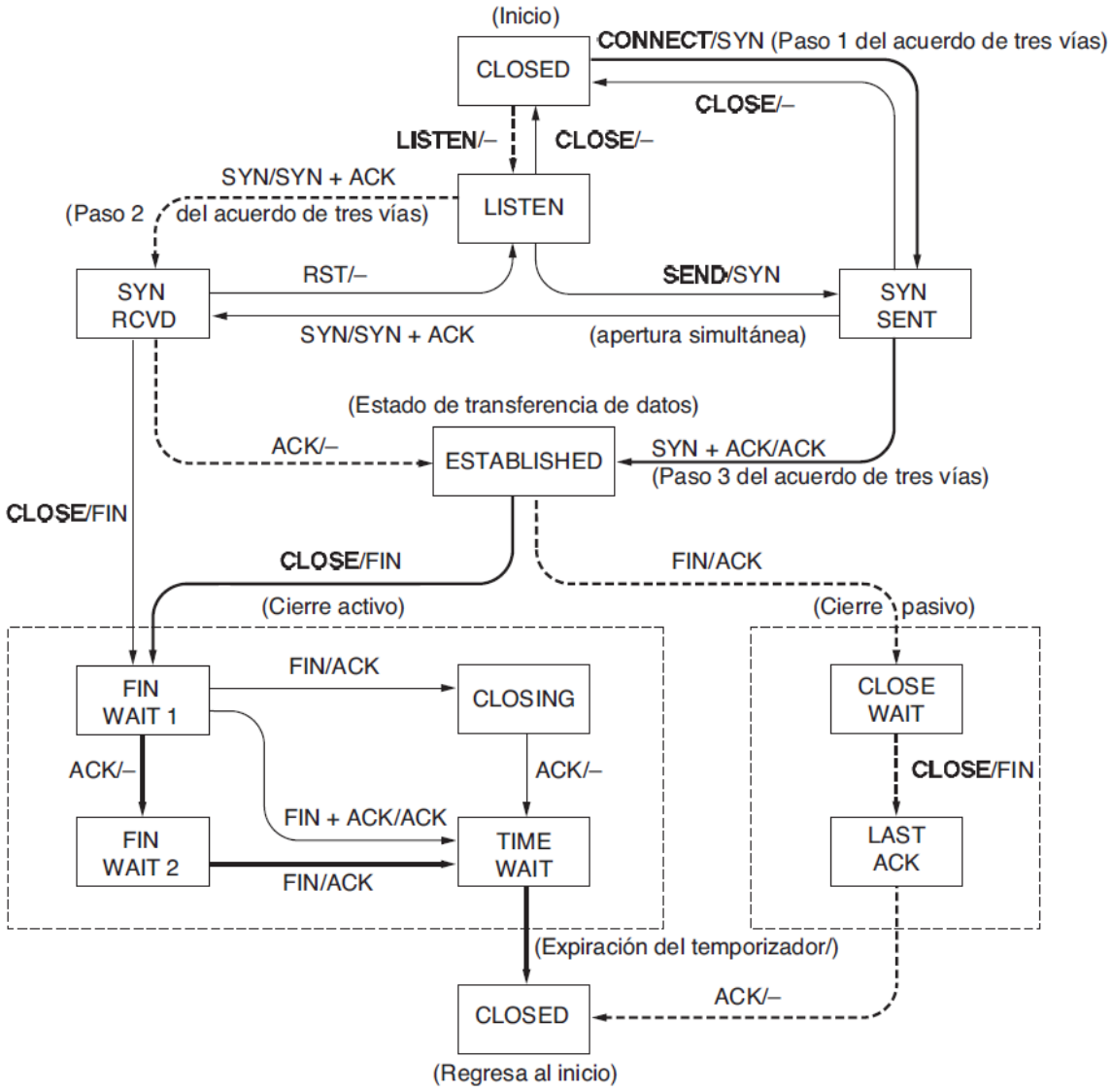


Figura 3.5: Máquina de estados de TCP [20]

El diagrama de flujo definido para la entidad emisora (figura 3.5 marcado con línea continua) comienza con la *fase de establecimiento*, la cual se inicia con el envío de un paquete *TCP-SYN* y la ejecución del proceso de '*CONNECT*'. Una vez finalizados los dos procesos (envío de la trama y '*CONNECT*') la máquina de estados local de TCP pasa al estado '*SYN SENT*', es decir, indica al sistema que se ha enviado el paquete *TCP-SYN* y está esperando a la recepción de la confirmación por parte de la entidad receptora. El sistema emisor permanece en este estado hasta recibir un paquete '*TCP-SYN+ACK*' procedente del sistema receptor. Una vez el emisor recibe dicho paquete cambia al estado de conexión establecida o '*ESTABLISHED*'. En este estado el emisor realiza todas las transmisiones y/o recepciones necesarias para la comunicación entre las entidades. Para finalizar, el emisor debe llevar a cabo la *fase de cierre de conexión*. Este proceso de cierre, a diferencia del proceso de cierre de la entidad receptora, es un proceso donde el emisor indica al receptor que no tiene más información que transmitir pero que se queda a la

espera de la información o cierre por parte del receptor. El proceso de cierre comienza con el envío de un paquete *TCP-FIN* a la entidad receptora y con un cambio de estado, pasando del estado '*ESTABLISHED*' al estado '*FIN WAIT 1*'. Este estado permanece activo hasta la llegada de un paquete *TCP-ACK* desde el receptor, tras lo cual, el cliente pasa a un estado '*FIN WAIT 2*', cerrando así el sentido emisor-cliente de la comunicación.

Por su parte, el receptor inicia su máquina de estados (figura 3.5 dibujado con línea discontinua) mediante la activación del proceso '*LISTEN*', el cual permite al receptor escuchar posibles conexiones. Una vez el receptor recibe un paquete *TCP-SYN* solicitando una conexión, contesta mediante el envío de un paquete *TCP-SYN+ACK* y avanza al estado '*SYN RCVD*' a la espera de confirmar la conexión mediante la llegada de un paquete *TCP-ACK* procedente de la entidad emisora. Una vez dicho paquete es recibido por el receptor, éste cambia al estado de conexión establecida o '*ESTABLISHED*', donde al igual que el emisor, intercambia los diferentes mensajes para realizar la transferencia de información. Al contrario que la entidad emisora, la entidad receptora comienza el proceso de cierre de conexión únicamente cuando recibe por parte del emisor un paquete *TCP-FIN*. Una vez recibido dicho paquete, el sistema receptor contesta con un paquete *TCP-FIN* al emisor y procede al borrado de los diferentes registros para así ejecutar el proceso de cierre de la conexión.

Otra de las características más importantes del protocolo TCP es la provisión de un control de errores [21] extremo a extremo. Este mecanismo se basa en confirmaciones acumulativas, es decir, una vez recibido una confirmación, el sistema supone que todos los paquetes pendientes de confirmación recibidos con anterioridad al confirmado también son correctos de forma implícita (indirecta). Para ello, TCP dispone de cuatro campos dentro de la cabecera de cada uno de los paquetes. Estos campos permiten comprobar si se han recibido todos los paquetes, para en el caso de faltar alguno, solicitar su retransmisión. Los campos necesarios para llevar a cabo el mecanismo anterior son: *el número de secuencia*, el cual hace referencia a la posición que ocupa el primer octeto del campo de datos transportado en el paquete con respecto al total de octetos que componen el flujo total; *el checksum* o suma de comprobación; *el número de asentimiento*, indica el número de secuencia del siguiente octeto que se espera recibir; y por último, el flag de *ACK* que indica la validez de la confirmación.

TCP también provee a la comunicación de un control de flujo [21]. Este control de flujo se ha implementado utilizando el concepto de '*ventanas deslizantes*' y hace uso del campo *ventana* de la cabecera TCP para su control. Este campo indica el número de octetos que se autoriza a enviar de forma consecutiva. Por lo tanto, en una comunicación el receptor especifica al emisor su tamaño de ventana máximo, para que éste transmita como máximo dicho número de octetos, tomando como emisor en este caso a la entidad que vaya a proceder a la transmisión de información y no al sistema que haya iniciado el proceso de comunicación.

Por último, *TCP* dispone de un sistema para controlar la congestión de la red extremo a extremo. Este mecanismo pretende limitar la velocidad a la que el emisor puede transmitir los paquetes sin que ello cause una congestión en la red. Para ello, el emisor calcula el número de octetos que puede transmitir sin que exista confirmación por parte del receptor igualando dicho número de octetos al valor más pequeño dado por las *ventanas de recepción* (control de flujo) y por la *ventana de congestión*, esta ventana establece la cantidad máxima de octetos que se pueden trasmitir de forma consecutiva a partir de la percepción que tiene TCP de la congestión de la red. Sin embargo, una duda que surge al conocer el significado de las diferentes ventanas es: ¿cómo puede el número de octetos limitar la *velocidad de transmisión* de una comunicación?. Esto se debe a que al establecer una cota superior se limita la cantidad de información que puede estar sin confirmar en una transmisión, lo cual limita la velocidad en que se generan nuevos paquetes y, por lo tanto, limita la velocidad en la transmisión de los datos [27]. Actualmente, existen diversos algoritmos de control de congestión utilizados por TCP, a continuación, se van a explicar algunos de ellos.

Algoritmo de control de congestión RENO

Reno [27] es el mecanismo clásico de control de congestión y sigue la máquina de estados representada en la figura 3.6 [27].

En ella se observa la existencia de tres grandes estados: *Arranque Lento*, *Evitación de la Congestión* y *Recuperación Rápida*.

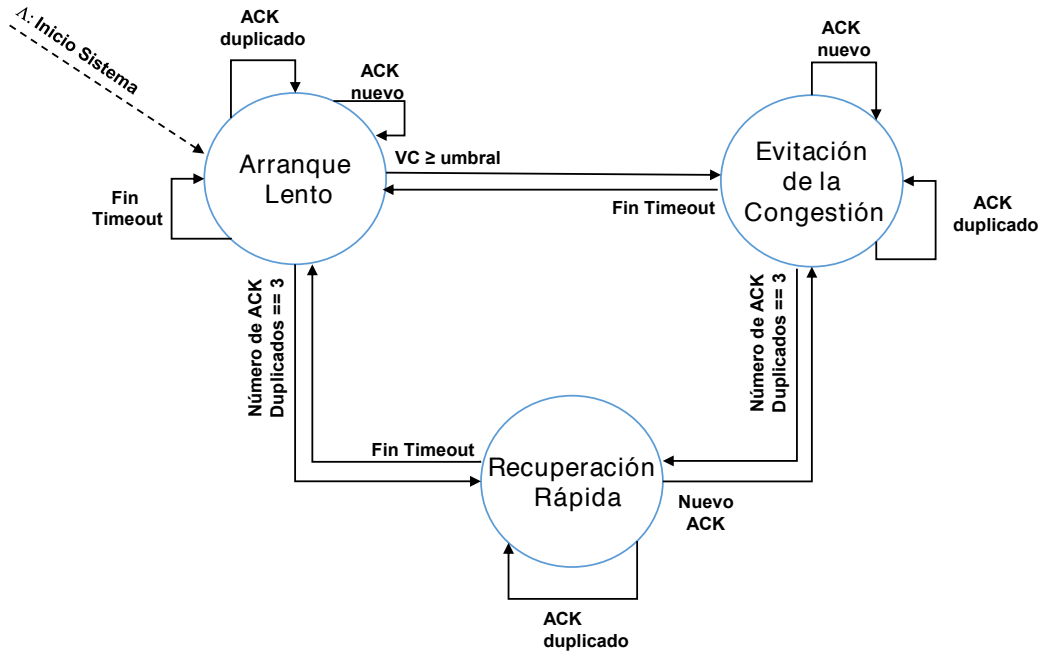


Figura 3.6: Máquina estados del algoritmo de control de congestión Reno

- *Arranque Lento*: al comienzo de este estado se inicializa el valor del *Umbral* igualándolo a 64 Kbyte, el *Umbral* es un parámetro utilizado por TCP para estimar el momento desde el cual puede existir riesgo de congestión. A continuación, se establece el valor del tamaño de la *Variable de Control de la Ventana de Congestión (Vc)*, dicho valor ha variado con el tiempo desde el asignado por las primeras implementaciones donde el valor se igualaba al mismo valor establecido para el *Maximum Segment Size (MSS)* hasta el utilizado en desarrollos posteriores donde se le asigna el doble del valor de la MSS. Por último, durante esta fase se va incrementando la ventana de congestión en una MSS por cada *Acknowledgement Packet (ACK)* no duplicado que reciba el sistema. Esta fase finaliza en los siguientes casos: si el sistema emisor no recibe un ACK en un tiempo determinado, lo que provoca que el valor del *Umbral* se reduzca a la mitad, el valor de Vc se iguale a 1 MSS y se reinicie la fase de *Arranque Lento*; o se reciban tres ACK duplicados, lo que produce que el valor del *Umbral* se reduzca a la mitad, el valor de Vc se iguale al valor del nuevo *Umbral* más tres veces el valor de la MSS ($Vc = Umbral + 3 * MSS$) y se pase a la fase de *Recuperación Rápida*; o, por último, si el valor de la Vc se iguala al valor del *Umbral* se pasa a la fase de *Evitación de la Congestión*.

- *Evitación de la Congestión*: al estar 'cerca' de una posible congestión, el sistema establece un aumento de Vc mucho más conservador. El nuevo valor de Vc se calcula mediante la suma: $Vc_{Nueva} = Vc_{anterior} + \frac{MSS}{Vc_{anterior}}$ por cada ACK no repetido recibido. Esta fase finaliza, bien porque no se ha recibido un ACK, lo que provoca que el valor del *Umbral* pase a la mitad de su valor anterior, el valor de Vc se iguale al valor de 1 MSS y se pase al estado de *Arranque Lento*, bien porque el sistema reciba tres ACK duplicados, lo que provoca que el sistema divida a la mitad el valor del *Umbral* y, dependiendo de la implementación, disminuya Vc a $Vc = Umbral + 3 * MSS$ y se pase al estado de *Recuperación Rápida*; o iguale el valor de Vc al de MSS y se pase al modo *Arranque Lento*.

- *Recuperación Rápida*: en esta fase la ventana de congestión se incrementa en un MSS por cada ACK duplicado que se ha recibido. Esta fase acaba: bien, cuando se recibe el ACK del paquete perdido y, por lo tanto, el sistema iguala el valor del *Umbral* a la mitad del valor de Vc anterior a la detección de la congestión, asimismo, se iguala la nueva Vc al valor del nuevo *Umbral* y pasa al estado de *Evitación de la Congestión*. Si, no se recibe el ACK del paquete perdido dentro del tiempo definido, provoca que el valor del *Umbral* pase a valer la mitad del valor de Vc anterior a la detección de la congestión, el valor del Vc se iguala con el valor de MSS y se pasa al modo *Arranque Lento*.

Un ejemplo de la evolución de la ventana de congestión de *Reno* se puede observar en la figura 3.7 [27].

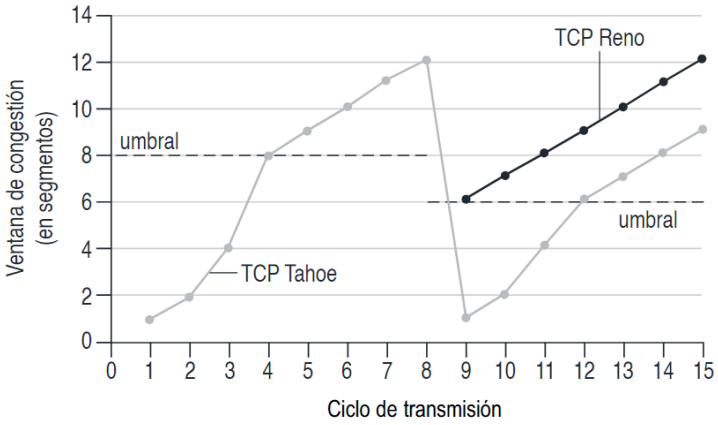


Figura 3.7: Representación de la evolución de la ventana de congestión de Reno [27]

Algoritmo de control de congestión NEW-RENO

El algoritmo de control de congestión establecido por *NEW-RENO* [28] es el resultado de una modificación en el algoritmo definido por *RENO*. La gran diferencia entre ambos algoritmos reside en que *NEW-RENO* permite realizar retransmisiones en paralelo debido a la pérdida de múltiples paquetes.

NEW-RENO al igual que su predecesor (*RENO*) entra en la fase de *Recuperación Rápida* al detectar la pérdida de paquetes mediante la llegada de paquetes *TCP-ACK* duplicados, pero a diferencia de *RENO*, *NEW-RENO* solo abandona esta fase cuando recibe *TCP-ACK* para todos los paquetes perdidos, evitando así, una reducción del tamaño de su Vc múltiples veces.

Algoritmo de control de congestión VEGAS

La implementación del Algoritmo de *control de congestión* realizado en *Vegas* [29] se diseña estableciendo como hipótesis que una medida proactiva es mejor que una medida reactiva. Esta hipótesis se observa en la manera que tiene *Vegas* de definir el tamaño de Vc, ya que a diferencia de *Reno*, *Vegas* predice la posible congestión de una red calculando el valor del Round-Trip Time (RTT) de los ACK. Por lo tanto, cuando *Vegas* detecta un aumento en el RTT, interpreta que existe un posible aumento de la congestión, motivo por el cual disminuye el valor de Vc de forma proactiva. Por el contrario, si el RTT disminuye, *Vegas* estima que el valor de la posible congestión ha bajado y, por lo tanto, aumenta el valor de Vc. *Vegas* en su fase de *Arranque Lento*, a diferencia de *Reno*, realiza el incremento de Vc únicamente al final de cada *RTT*.

Estas diferencias se muestran en la figura 3.8 [30]. En ella se observa como las variaciones en Vc de *Vegas* se producen antes que las de *Reno*, ya que aplica un algoritmo que predice la posible congestión existente en la red, calculando para ello el valor del RTT de los ACKs, mientras que *Reno* únicamente se basa en el evento producido por la recepción de los ACKs para realizar la modificación de Vc.

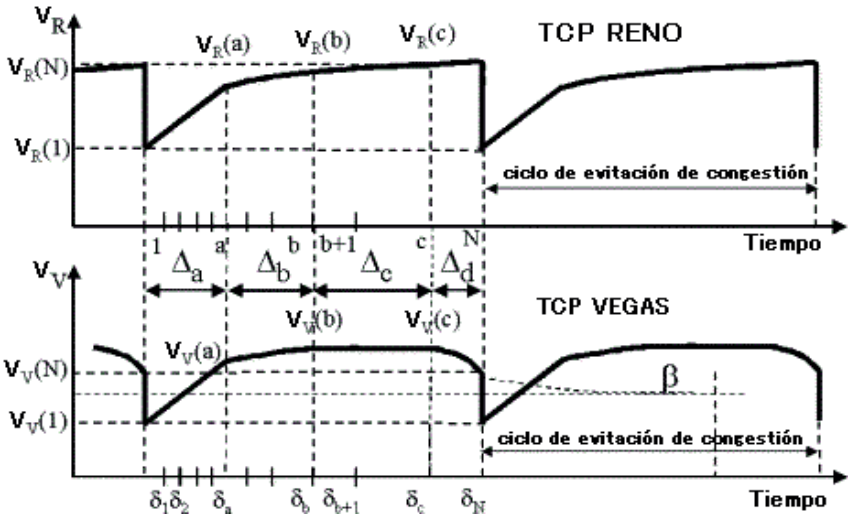


Figura 3.8: Comparación de la evolución de las ventanas de congestión de Reno y Vegas [30]

Algoritmo de control de congestión BIC

El siguiente algoritmo de control de congestión es Binary Increase Congestion control (BIC). Este algoritmo fue definido por K. Lisong Xu y compañía en [31]. Los autores se propusieron como objetivo principal para su desarrollo la optimización de las Vcs en las 'nuevas redes de alta velocidad y grandes RTTs' donde los mecanismos anteriores dejaban de funcionar de forma eficiente. Para ello, los autores definieron para BIC un algoritmo de control de congestión capaz de obtener un valor más óptimo de Vc en función de la disponibilidad en la red tomando para ello como base el diagrama de estados definido en *New Reno* e introduciendo en él un nuevo estado (*Fast Convergence*). El diagrama de estados se estableció con los siguientes estados:

- *Arranque Lento* [31]: la función principal de esta fase es la de establecer el valor

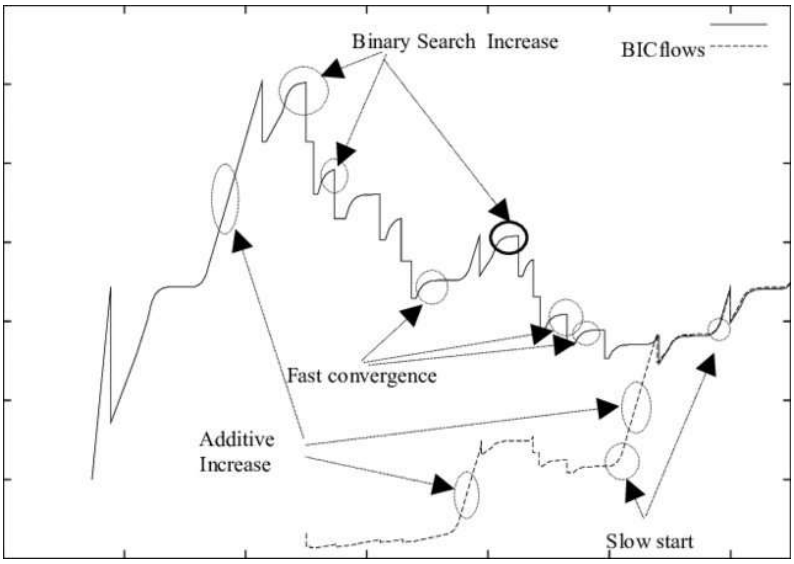


Figura 3.9: Ejemplo de evolución de una ventana de congestión aplicando BIC [31]

máximo que puede tener Vc, también conocido como Valor Máximo de la Variable de Control de la Ventana de Congestión (Vcmax). El nuevo valor de Vcmax se calcula mediante incrementos, es decir, mediante la suma de Vcmax y el producto entre el número de RTTs y el valor del Incremento Mínimo en el Sistema (Smin) definido en el sistema. Por ejemplo, para un único RTT el valor de Vcmax sería $Vcmax = Vcmax + 1 * Smin$, para dos RTTs se tendría un valor igual a $Vcmax = Vcmax + 2 * Smin$, así sucesivamente hasta que el valor obtenido por dicha suma ($Vcmax = Vcmax + N * Smin$ siendo N el número de RTTs) se sitúe por encima del valor obtenido a partir de la suma de los valores de Vcmax y del Valor del Incremento Máximo en el Sistema (Smax) ($Vcmax + Smax$). Este efecto introducido por esta fase sobre la Vcmax se observa en los puntos denominados *Slow Start* situados dentro de la figura 3.9 [31].

- *Binary Search Increase* [31]: esta fase tiene como principales objetivos la obtención del valor medio entre los valores Vcmax y Valor Mínimo de la Variable de Control de la Ventana de Congestión (Vcmin), y el ajuste de Vc. En esta fase, el sistema comienza calculando el valor medio mediante la ecuación $\frac{Vcmax-Vcmin}{2}$ para una vez obtenido dicho valor continuar con la actualización de los valores asociados a Vcmax y Vcmin. Esta actualización tiene en cuenta la existencia o no de paquetes perdidos, es decir, si existe pérdida de paquetes entonces se asigna a Vcmax el valor del punto medio calculado ($\frac{Vcmax-Vcmin}{2}$). En caso contrario, es a Vcmin a la que se le asigna el valor de dicho punto. Se puede observar la evolución de la ventana durante esta fase en las zonas marcadas como '*Binary Search Increase*' dentro de la figura 3.9 [31].

- *Additive Increase* [31]: esta fase establece que, si la distancia entre el punto medio entre Vcmax y Vcmin ($\frac{Vcmax-Vcmin}{2}$) y el valor de Vcmax actual es muy grande, actualizar el valor de Vcmax a dicho punto puede llegar a ser un problema para el sistema. Para evitarlo, esta fase define el parámetro (Smax) con el objetivo de igualar el valor de Vcmax al valor calculado mediante sucesivas sumas de aproximación hasta que el resultado de la suma supere al valor medio obtenido en la fase de *Binary Search Increase*, es decir, aplicando un proceso iterativo muy similar al ejecutado durante la fase de *Arranque Lento*, pero esta vez, las sumas se realizan siguiendo la expresión: ($Vcmax = Vcmax + Smax$). Para comprender mejor el efecto de esta fase sobre el tamaño de Vc se han de observar las zonas marcadas como '*Additive Increase*' sobre la figura 3.9 [31].

- *Fast Convergence* [31]: esta fase establece dos posibles casos, el primero dice que existiendo una pérdida de paquetes y, por lo tanto, una bajada en el valor de Vc, el sistema toma $\frac{Vcmax-Vcmin}{2}$ como nuevo valor de Vc; mientras que, si el sistema detecta un incremento en el valor de Vc, asigna el valor de Vcmax como nuevo valor de Vc. La figura 3.9 [31] muestra un ejemplo de cómo afecta esta fase a la ventana de congestión del sistema.

Algoritmo de control de congestión CUBIC

El último algoritmo de control de congestión que se va a presentar en esta tesis es *CUBIC*. Aprovechando el aumento en el ancho de banda de las actuales líneas de comunicación, *CUBIC* pretende definir un algoritmo de control de congestión para TCP mucho más agresivo, para ello basa su diseño en el uso de una función cúbica del tiempo que transcurre desde el instante del último evento hasta el momento actual [32].

$$Vc = C\left(\Delta - \sqrt[3]{\frac{\beta * Vcper}{C}}\right)^3 + Vcper \tag{3.1}$$

La función cúbica que define el comportamiento de *CUBIC* sigue la expresión matemática: 3.1, donde C es un valor definido directamente en el sistema, β es una variable cuyo valor se establece en cada *fase de recuperación rápida* y Valor de la Variable de Control de la Ventana de Congestión Antes de una Pérdida (Vcper) es el valor de Vc justo antes de detectar la pérdida de un paquete.

El funcionamiento de *CUBIC* se puede describir siguiendo la figura 3.10 [32]. La primera fase, representada como '1' en dicha figura, se establece para conocer el valor de la Vcper. Para ello, el sistema aplica un incremento de la Vc siguiendo la parte cóncava de la función matemática (viendo la representación de la figura 3.11 [32] coincide con la denominada como '*Max Probing*'). Esta parte de la función propone un aumento muy agresivo de la ventana de congestión consiguiendo así llegar a valores muy altos en un corto espacio de tiempo.

Por otro lado, la fase indicada como '2' dentro de la figura 3.10 [32] se inicia en el momento en que el sistema detecta la pérdida de un paquete, reduciendo Vc e iniciando su incremento siguiendo el lado convexo de la función (viendo la representación de la figura 3.11 [32] coincide con el denominado '*Steady State Behavior*'). Esta fase propone un aumento mucho más moderado que la fase anterior ya que al detectarse una pérdida de paquetes el sistema lo interpreta como una posible congestión en la red. Esta fase finaliza

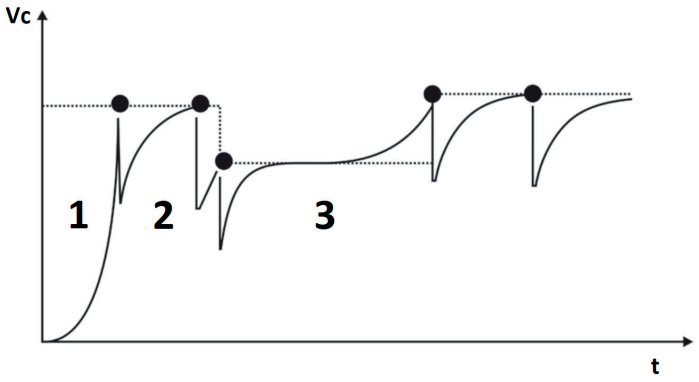


Figura 3.10: Ejemplo de evolución de una ventana de congestión aplicando CUBIC [32]

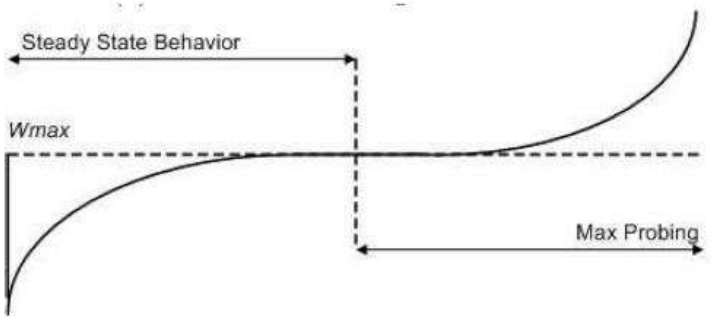


Figura 3.11: Evolución de la función cúbica empleada por la ventana de congestión aplicando CUBIC

en dos posibles situaciones. La primera cuando se detecta una nueva pérdida de paquetes, en este caso se reinicia la fase, o por el contrario cuando el valor de Vc sobrepasa el valor de Vcper, lo que provoca que el sistema realice de nuevo la fase '1'. Este cambio se ha representado en la figura 3.11 [32] como '3'.

En resumen, se puede ver como *CUBIC* introduce modificaciones de Vc elevadas cuando Vc es muy inferior o superior al punto de inflexión de la función cúbica. Por el contrario, si Vc se encuentra próximo al punto de inflexión, *CUBIC* establece unas modificaciones menos agresivas ya que la diferencia entre Vc y el punto de inflexión es menor.

3.1.3.3 Comparación UDP vs TCP

Tras conocer los dos protocolos de transporte, a modo de resumen, se presenta una tabla que compara las propiedades más relevantes de cada uno de ellos (ver tabla 3.1).

3.2 SDN: Redes definidas por software

La arquitectura SDN [33], define una arquitectura que rompe con las arquitecturas tradicionales ya que separa el plano de datos del *Plano de Control*, no solo de manera lógica dentro de una misma entidad, sino también físicamente. Además, tiene como uno de sus principales objetivos mejorar y facilitar tanto la gestión como la administración de las redes y sus servicios mediante una abstracción de las funciones de bajo nivel. Por otro lado,

TCP	UDP
Es un servicio orientado a conexión.	Es un servicio no orientado a conexión.
TCP garantiza la entrega de los paquetes mediante el uso de confirmaciones.	No garantiza que los paquetes lleguen a su destino.
Es un servicio pesado, complejo y permite únicamente conexiones entre par de equipos finales.	Es un servicio ligero, más sencillo permite transferencia punto a multipunto.
Dispone de sistemas de control de flujo, ordenación de paquetes y control de congestión.	No dispone de control de flujo.
TCP dispone de sistemas de verificación de errores y dispone de mecanismos para corregir dichos errores.	UDP dispone de comprobación de errores pero no dispone de mecanismos para corregir los errores.
Dispone de multiplexación en un mismo equipo.	Dispone de multiplexación en un mismo equipo.

Tabla 3.1: Comparación de las características principales de TCP y UDP

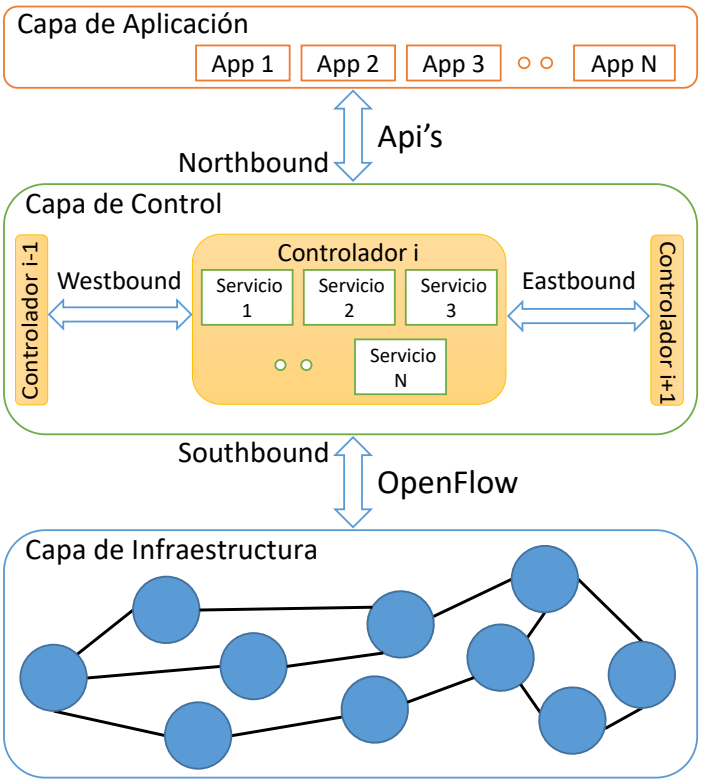


Figura 3.12: Arquitectura SDN

la arquitectura SDN también intenta hacer más accesible la creación, implementación e implantación de nuevos servicios de red, introduciendo para ello una capa intermedia que provea a los desarrolladores de un conjunto de funciones de red.

3.2.1 La arquitectura SDN

La arquitectura SDN separa el *Plano de Control* del plano de datos dentro de los diferentes switches de la red, rompiendo así con lo establecido por las arquitecturas tradicionales durante años. Para ello, elimina la capacidad de toma de decisiones de los switches de la red y se la entrega a una entidad superior denominada *Controlador*. A su vez, establece una separación de la arquitectura de red siguiendo un modelo de capas (ver figura 3.12 [11]).

Este nuevo modelo se compone de tres capas diferentes:

- *Capa de Aplicación*: es la capa [34] situada en la parte superior del modelo, y en ella, los gestores de red establecen, desarrollan, definen e implementan los diferentes módulos y aplicaciones que controlan y administran la red.

- *Capa de Control*: es el centro del modelo definido por el paradigma SDN, se la conoce comúnmente como *Controlador* y se puede equiparar funcionalmente a un *middleware*. Esta capa provee a la capa de superior de un conjunto de funciones que permiten a los desarrolladores entre otras cosas implementar sus aplicaciones,

comunicarse con la capa inferior, comunicarse con otros controladores, etc. Además, es el encargado de codificar y transmitir las 'ordenes' a los switches de la red.

- *Capa de Infraestructura*: es la capa hardware del modelo [33], está formada por todos los switches capaces de establecer una comunicación con el controlador.

Esta arquitectura también define las interfaces o mecanismos de comunicación entre capas: las *Capas de Aplicación y de Control* se comunican a través de una Application Programming Interface (API) denominada *Northbound API*. Esta API proporciona a los desarrolladores las funciones necesarias para comunicarse con la Capa de Control de una forma sencilla. Por otra parte, la *Capa de Control* y la *Capa de Infraestructura* necesitan de la interfaz para poder comunicarse, esta interfaz se ha denominado como *Southbound Interface* y se encarga de proporcionar los mecanismos, acciones y protocolos necesarios para llevar a cabo la comunicación entre cada uno de los switches y la *Capa de Control*. Cabe citar los dos protocolos más utilizados para llevar a cabo la comunicación: *OpenFlow* [35] y *GRPC* [36]. Adicionalmente algunos controladores disponen de interfaces de conexión entre controladores, estas interfaces se denominan *Westbound API y Eastbound API* y su principal función es aumentar la escalabilidad y disponibilidad del *Plano de Control* mediante el uso de varios controladores SDN.

3.2.2 Protocolo OpenFlow

La comunicación entre cada uno de los dispositivos que forman la *Capa de Infraestructura* con la *Capa de Control* necesita de un protocolo de comunicación; por esta razón nació *OpenFlow* [37], el cual define un conjunto de normas, paquetes y procesos para llevar a cabo correctamente dicha comunicación. Asimismo, también define las características que deben cumplir cada uno de los switches SDN para su correcta integración en una red *SDN/OpenFlow*.

3.2.2.1 OpenFlow Pipeline

Uno de los puntos más delicados de este protocolo es el proceso que se va a realizar sobre cada uno de los paquetes entrantes en los switches *OpenFlow*, este proceso se denomina *Pipeline*.

El *Pipeline* (ver figura 3.13) hace uso de un conjunto de tablas ordenadas secuencialmente en cascada, donde el controlador puede crear nuevas reglas, modificar las ya existentes o simplemente consultarlas. El procesamiento que un switch *OpenFlow* realiza con un paquete entrante se basa en ir comparando de forma secuencial cada una de las reglas almacenadas en cada una de las tablas e ir guardando todas las acciones asociadas a cada una de las reglas donde se haya producido un *match*, para al final, ejecutar todas esas acciones en orden de aparición.

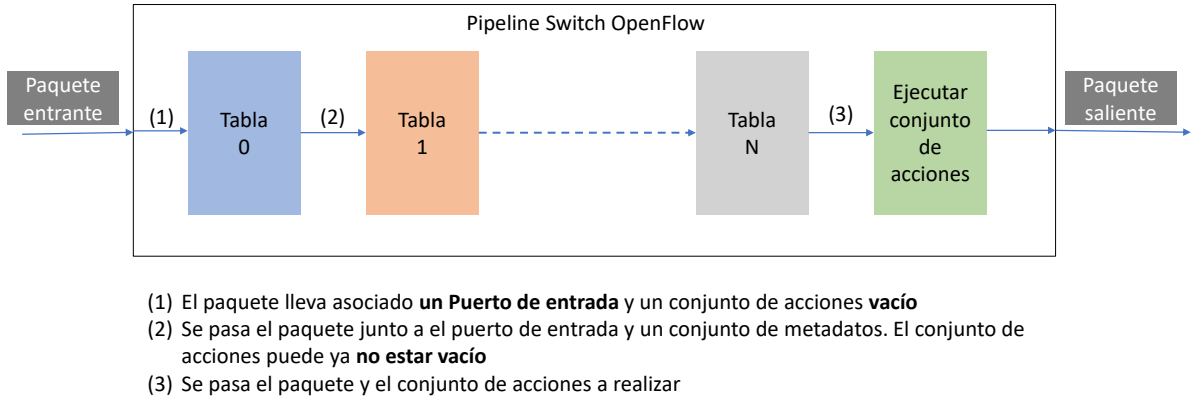


Figura 3.13: OpenFlow pipeline

En resumen, el paquete recibido en un switch *OpenFlow* se envía a cada una de las tablas del dispositivo que dispongan de al menos una regla en su interior (condición para considerar una tabla activa) y cada vez que se localice una regla que coincida con el paquete, se añade la acción establecida en la regla al final del conjunto de acciones a ejecutar, para al final tras pasar por todas las tablas llevar a cabo dichas acciones en orden de aparición.

3.2.2.2 Tablas OpenFlow

Las tablas definidas por el protocolo *OpenFlow*, como se ha podido ver hasta ahora, son uno de los elementos fundamentales para el correcto funcionamiento del protocolo. En ellas se almacena la información relacionada con los flujos (*Tabla de Flujo*), la información de grupos (*Tabla de Grupos*) y las medidas (*Tabla de Medidas*) [38].

La *Tabla de Flujos* tiene como función principal guardar las diferentes reglas que establecen cada uno los procesos, tratamientos y/o cambios a realizar sobre los diferentes paquetes recibidos. La estructura de esta tabla se representa en la tabla 3.2 [35].

OpenFlow establece que cada uno de los elementos que contiene la *Tabla de Flujos* debe tener los siguientes parámetros:

- *Campos de coincidencias*: son los parámetros gracias a los cuales se realizan las comparaciones con las cabeceras de los paquetes. Estos parámetros pueden ser tales como la dirección Media Access Control (MAC) origen, la dirección MAC destino, el puerto de entrada, la dirección IP tanto origen como destino e incluso algunos metadatos definidos por el propio protocolo *OpenFlow* o por el propio desarrollador.

Campos de *Match*	Prioridad	Contadores	Instrucciones	*Timeouts*	*Cookie*

Tabla 3.2: Definición de Tabla de Flujos (*Flow Table*)

Campos de Cabecera	Contadores	Acciones	Prioridad
Si puerto de entrada == 3	–	Tirar el mensaje	7585
Si IP Origen == 192.168.35.15	–	Actualizar el valor por 10.1.2.31 y reenviar por 4	3572
Si MAC Destino == AD:E2:3E:41:34:56	–	Add VLAN 12 y reenviar puerto 1	4214
Si puerto de entrada == 7	–	Reenviar por puertos 1,12	2314
Si EtherType == 0x0806	–	Encaminar hacia el controlador	4000

Tabla 3.3: Ejemplo Tabla de Flujos (*Flow Table*)

- *Contadores*: son valores estadísticos referentes a los diferentes paquetes o flujos procesados por el switch. Como por ejemplo el número de veces que una regla ha tenido una comparación positiva.

- *Instrucciones*: define los procesos o acciones a realizar después de que haya una coincidencia (*'match'* en terminología *OpenFlow*) con alguna regla establecida.

Para facilitar la compresión del funcionamiento de la *Tabla de Flujos* de *OpenFlow* se muestra un ejemplo en la tabla 3.3 [39]). En él se puede observar un conjunto de reglas almacenadas por el controlador dentro de la *Tabla de Flujos* de un switch. En dicho ejemplo se puede ver como una de esas reglas le está indicando al switch que todo paquete que le entre por el puerto número '3' debe ser eliminado, o como otra le indica que si el *EtherType* coincide con el del protocolo Address Resolution Protocol (ARP) entonces debe enviar dicha trama al controlador para su procesamiento.

Por otro lado, agrupar información permite a los sistemas reducir la utilización de memoria, por ello *OpenFlow* establece la necesidad de tener una *Tabla de Grupos*. Esta tabla es la estructura en la que se almacenan todas las reglas que definen el comportamiento de un conjunto de elementos asociados a un grupo, de tal manera, que esta información permite al controlador agrupar a todos aquellos elementos que realicen las mismas acciones bajo unas mismas condiciones. El ejemplo más claro para comprender el uso de la *Tabla de Grupos* es el reenvío o tratamiento de paquetes *Multicast*. La estructura establecida por el protocolo *OpenFlow* para esta tabla se representa en la tabla 3.4 [35].

Los registros guardados en la *Tabla de Grupos* están compuestos por los siguientes parámetros:

- *Identificador de Grupo*: Es un campo numérico que define de forma unívoca a cada uno de los grupos de la tabla.

- *Tipo de Grupo*: Define el tipo del conjunto. Existen diferentes tipos dentro del protocolo *OpenFlow* tres de ellos obligatorios (*Todos*, *Seleccionado* e *Indirecto*) y uno opcional (*Conmutación rápida tras un error* (*fast failover*)). El primero denominado como *Todos* establece que todos los elementos del grupo realizan todas las acciones

Identificador de Grupo	Tipo de Grupo	Contadores	Conjunto de Acciones

Tabla 3.4: Definición Tabla de Grupos (*Group Table*)

Identificador de Grupo	Tipo de Grupo	Contadores	Conjunto de Acciones
5	–	–	Puertos de salida x,y
15	–	–	Puerto de salida a
17	–	–	ID de Grupo D
6	–	–	Puerto de salida B e ID Grupo C
9	–	–	Tirar paquete

Tabla 3.5: Ejemplo Tabla de Grupos (*Group Table*)

definidas para los grupos, es muy utilizado para paquetes *multicast* o *broadcast*. El segundo denominado como *Seleccionado*, en este caso solo ejecuta las acciones asignadas a los grupos. El tercero se denomina *Indirecto* y es muy parecido al grupo anterior, pero con la diferencia de que la acción a realizar solo está dentro del conjunto de acciones posibles para el grupo en cuestión. Por último, el tipo denominado *Conmutación rápida tras un error* (*fast failover*), aunque opcional, es capaz de modificar el reenvío de los paquetes sin necesidad de consultar con el controlador.

- *Contadores*: se trata de datos numéricos utilizados principalmente para el cálculo de estadísticas.
- *Acciones asociadas*: se trata del conjunto de acciones y procedimientos que se deben realizar sobre los paquetes que coincidan con las reglas del grupo.

Como ejemplo de una tabla de grupos tenemos la tabla 3.5 [39]. En ella se muestran un conjunto de reglas introducidas por el controlador dentro de la *Tabla de Grupos* del nodo. En ella se observa como, por ejemplo, al grupo de elementos asociados al identificador 15 se les va a asignar como puerto de salida el 'a'.

Por último, la *Tabla de Medidas* almacena datos para realizar un control sobre los paquetes entrantes. Esta tabla permite, entre otras cosas, realizar operaciones de calidad de servicio y configuración de servicios de red personalizados. Su estructura se muestra en la tabla 3.6 [35].

3.2.2.3 Mensajes OpenFlow

Otro de los puntos más relevantes, y que no puede faltar, en la definición de cualquier protocolo de red son los mensajes de control que deben intercambiar los diferentes dispositivos que hagan uso del protocolo. En el caso del protocolo *OpenFlow* se definen tres tipos de mensajes distintos: los mensajes enviados siempre desde el controlador con destino un switch de la red (*Mensajes de Controlador a Switch*), los mensajes que se envían únicamente desde el switch hacia el controlador (*Mensajes Asíncronos*) y los mensajes que puede ser enviados por cualquiera de las dos entidades de la red (*Mensajes Simétricos*).

Identificador de la Medida	Lista de Medidas	Contadores

Tabla 3.6: Definición de Tabla de Medidas (*Meter Table*)

Mensajes de controlador a Switch

En este conjunto de mensajes la entidad que inicia la comunicación es el controlador y tienen como principal objetivo la consulta o configuración de reglas, opciones o estados de un switch *SDN/OpenFlow*. A diferencia del resto de mensajes, éstos no tienen por qué tener una respuesta por parte del switch, es decir, puede tratarse de un simple mensaje de configuración desde el controlador, por ejemplo, un mensaje para la instalación de una regla (*Mod_Flow*).

Mensajes Asíncronos

La comunicación para este conjunto de mensajes siempre se inicia por parte del switch *SDN/OpenFlow*, pero a diferencia del caso anterior siempre son mensajes tipo *petición-respuesta*. Sus funciones varían bastante de unos mensajes a otros, pero cabe destacar algunas de ellas: notificar modificaciones en el estado del switch o de la red, solicitar información para realizar acciones, etc.

Para comprender mejor este tipo mensaje se adjunta la figura 3.14 donde el switch *s1* recibe un paquete desconocido, procedente de *h1*, por lo que solicita mediante un mensaje *Packet_In* información al controlador para poder encaminar el paquete. El controlador una vez ha recibido dicho mensaje realiza las acciones o cálculos necesarios y envía la acción a realizar directamente sobre el paquete que originó la consulta (*Packet _Out*) y una regla donde establece todas las acciones que se deben hacer con todos aquellos paquetes que pertenezcan al mismo flujo de datos (*Flow_Mod*). Este mismo proceso se ejecuta cuando el switch *s1* recibe un paquete con destino desconocido y origen *h2*.

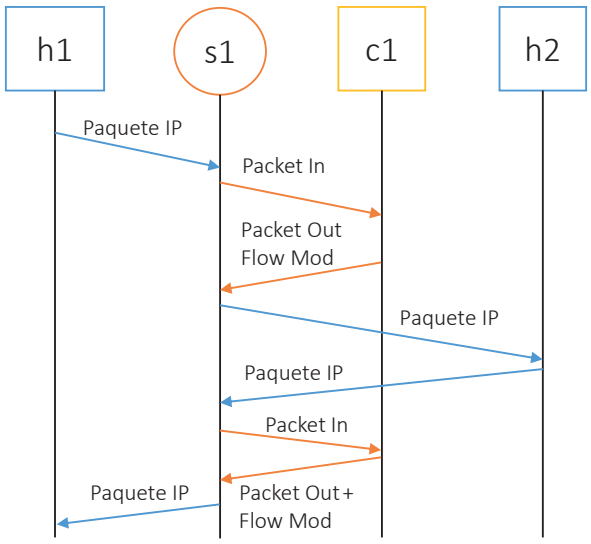


Figura 3.14: Intercambio mensajes asimétricos

Mensajes simétricos

En este grupo la comunicación puede ser iniciada tanto por el controlador como por el switch de red y al igual que en el grupo anterior estos mensajes son del tipo *petición-respuesta*. Este tipo de mensajes se dividen en dos grandes subgrupos, el primer subgrupo son los paquetes *Keep-alive*, es decir mensajes que indican al controlador que el switch está activo; mientras que el segundo subgrupo tienen como objetivo la obtención de información acerca del switch por parte del controlador, es decir, son los mensajes que permiten al controlador conocer las características (tanto físicas como lógicas) de los switches, o de sus puertos u obtener los valores estadísticos almacenados.

En la figura 3.15 se han representado tres de los mensajes más representativos de este conjunto. Primero el mensaje *Hello*, un mensaje clásico de *Keep-alive*. A continuación, el par de mensajes representados son los mensajes de *Feature_Request* y *Feature_Reply*, que permiten obtener al controlador las características físicas del switch. Por último, se representan los mensajes *Port_Desc_Stats_Request* y *Port_Desc_Stats_Reply*, gracias a los cuales el controlador puede conseguir la información relativa a los puertos e interfaces del switch.

3.2.3 Switch OpenFlow

El protocolo *OpenFlow*, como se ha mencionado en párrafos anteriores, define las características básicas que debe cumplir un dispositivo de red para ser considerado un switch *OpenFlow* y, por lo tanto, compatible con él.

Las características básicas establecidas para que un switch pueda considerarse compatible con el protocolo *OpenFlow* son las siguientes (ver figura 3.16). En primer lugar, debe disponer de una tabla donde el controlador pueda insertar reglas que permiten asociar una o varias acciones a un determinado tipo de paquetes, es decir debe disponer

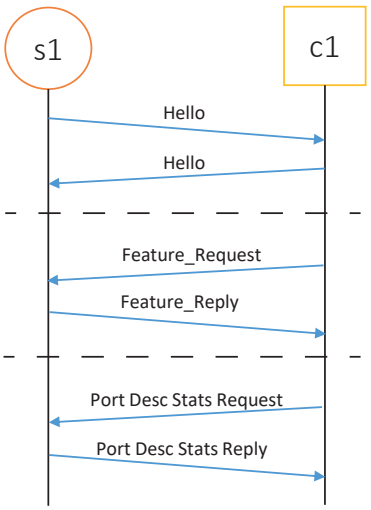


Figura 3.15: Intercambios de mensajes simétricos

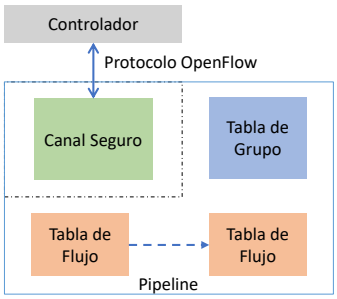


Figura 3.16: Funcionamiento del canal OpenFlow

de lo que *OpenFlow* denomina una *tabla de Flujos*. En segundo lugar, debe ser capaz de establecer una comunicación segura con el controlador, esta comunicación siempre debe ser sobre un canal TCP y podrá establecerse mediante mecanismos Transport Layer Security (TLS) para darle seguridad extra. Para terminar, el switch debe tener un lugar donde almacenar el conjunto de reglas definidas para grupos, o lo que es lo mismo, una *Tabla de Grupos*.

3.2.3.1 Lógica funcional de un switch OpenFlow

La figura 3.17 muestra el diagrama de flujo que representa el tratamiento OpenFlow 1.3 de las tramas recibidas en un nodo.

En la figura 3.17 se observa cómo, en primer lugar, el dispositivo aplica una función de análisis, lectura y almacenaje de los datos indicados en las cabeceras de la trama entrante como, por ejemplo, las 'direcciones MAC' (tanto origen como destino) de la cabecera de enlace, el campo 'tipo de protocolo' de la cabecera de red y los 'puertos origen y destino' de la cabecera de transporte. Estos datos se insertan en las estructuras internas del dispositivo para posteriormente, poder realizar las diferentes operaciones. En segundo

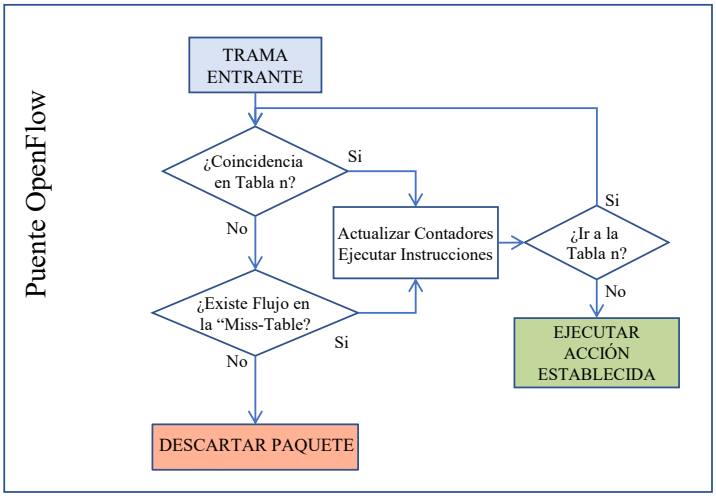


Figura 3.17: Diagrama de flujo funcional switch OpenFlow 1.3

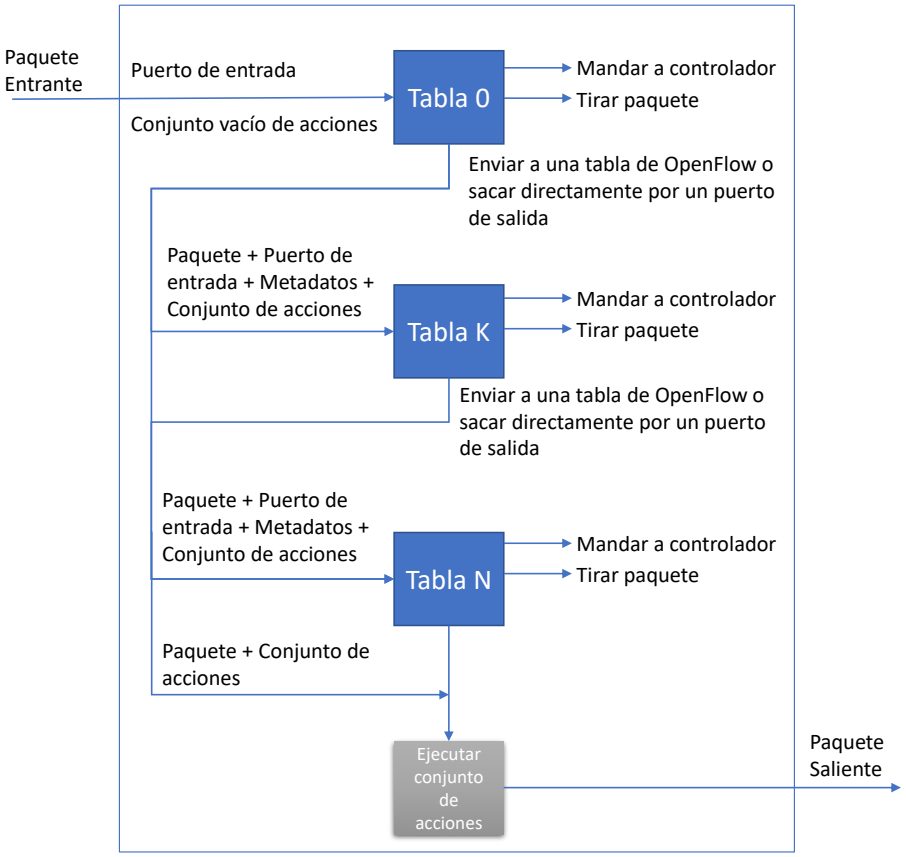


Figura 3.18: Procesado interno de una trama en un switch SDN

lugar, el dispositivo realiza una serie de comparaciones entre los datos obtenidos de las cabeceras de la trama y las reglas insertadas por el controlador en las diferentes tablas del dispositivo. Como existe la posibilidad de, por un lado, encontrar distintas reglas para un mismo tipo de paquete en la misma tabla y, por otro lado, de disponer de más de una tabla donde el controlador puede insertar sus reglas (ver figura 3.18), el dispositivo revisa cada una de las tablas de forma secuencial (tabla 0, tabla 1, tabla 2 ... tabla N-1, tabla N). En cada de una de ellas, revisa todas las reglas creadas en orden de prioridad. Si al realizar dicha comparación se encuentra alguna regla válida, se inserta la acción asociada a la regla en una lista de acciones a realizar con/sobre la trama. Sí, por el contrario, al finalizar el análisis el dispositivo no ha encontrado ninguna coincidencia y, por lo tanto, no dispone de ninguna acción en la lista, comienza a buscar en la tabla por defecto o '*Miss-Table*' (tabla donde el controlador debe introducir la acción por defecto a realizar) y, solo en caso de que tampoco se encuentre una acción válida, se descarta la trama.

3.2.4 Implementaciones de switches software compatibles con OpenFlow 1.3

Actualmente existen múltiples implementaciones software de dispositivos compatibles con el protocolo *OpenFlow* en su versión 1.3, en lo que respecta a esta tesis cabe citar dos de ellas. En primer lugar se desea citar *Basic OpenFlow Software Switch*

(BOFUSS) [15] inicialmente denominado OfSotfSwitch13 [40], se trata de un switch software implementado por el Centro de Investigación y Desarrollo de Telecomunicaciones (*CPqD*) de Brasil y que trata de cumplir fielmente con las especificaciones dadas por *OpenFlow* en su versión 1.3. En segundo lugar, Open vSwitch (OvS) [41], este switch software es el más utilizado debido a su altísimo rendimiento y su gran integración en plataformas hardware.

3.2.4.1 Basic OpenFlow Software Switch (BOFUSS)

BOFUSS [15] inicialmente denominado OfSoftSwitch13 [40], es un switch software completamente desarrollado en *espacio de usuario* de Linux. El lenguaje utilizado para su implementación es C/C++ y busca una implementación sencilla, accesible y cuyo objetivo principal es el desarrollo de un switch software lo más fiel posible a las características definidas por protocolo *OpenFlow 1.3*.

BOFUSS [15] está formado principalmente por 3 elementos: un bloque encargado de parsear o deserializar la trama, un bloque con la lógica del protocolo *OpenFlow* y las tablas donde almacenar información de grupos, flujos y/o puertos. La estructura interna de este software switch se resume en la figura 3.19 [35]. Esta figura muestra como *BOFUSS* comienza su flujo de trabajo cuando el switch recibe una trama a través de los puertos de entrada El sistema realiza llamadas periódicas a los *buffers* de recepción a través de '*Sockets RAW*'. En el caso de encontrar objetos en dichos *buffers* pasa al segundo bloque, donde en primer lugar la librería *NetBee Link* se encarga de llevar la trama desde la interfaz de red (Espacio de *Kernel*) al espacio de operaciones accesible por *BOFUSS* (Espacio de Usuario), para que en segundo lugar, *NetBee Parser* se encargue de *parsear*

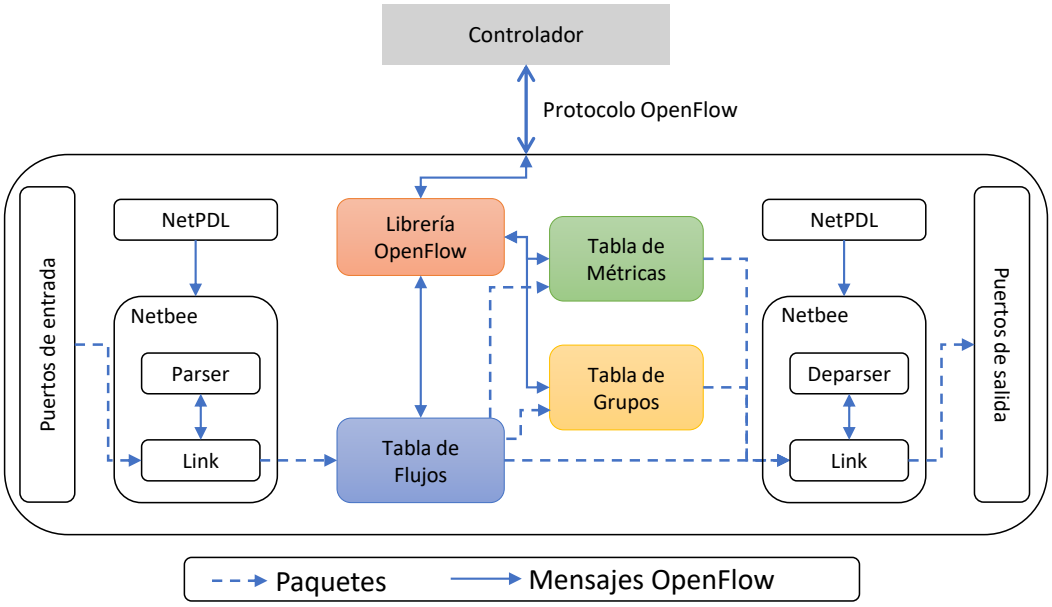


Figura 3.19: Arquitectura de BOFUSS

los datos procedentes de las cabeceras, es decir, se encarga de leer y deserializar los datos situados en las cabeceras de los paquetes e introducirlos dentro de las estructuras de datos del switch. A continuación, *BOFUSS* comienza un proceso de comparación entre dichos valores y las diferentes reglas guardadas dentro de la *Tabla de Flujos*, este proceso está controlado en todo momento por el bloque de control de *OpenFlow* (bloque denotado como *Oflib* en la figura, el cual también controla todos los procesos que tienen que ver con las *Tablas de Medidas* y con las *Tablas de Grupos*). Una vez finalizado el proceso de comparación el switch actualiza su *Tablas de Medidas* con los nuevos valores obtenidos del paquete procesado y, por último, si las hubiere, comienza con la realización de las acciones asociadas a las reglas, como, por ejemplo, reenviar el paquete por alguno de los puertos de salida del switch, actualizar algún dato de su cabecera o simplemente tirar el paquete.

Casos de uso de BOFUSS

BOFUSS es un switch que funciona únicamente en espacio de usuario y como se ha explicado en los apartados anteriores su programación y diseño son muy sencillos, lo que permite que la comunidad científica pueda reprogramarlo sin demasiado esfuerzo. Por lo que *BOFUSS* ha sido elegido para múltiples desarrollos y prototipados. De todos ellos, esta tesis quiere citar los siguientes:

- *BEhavioural BAsed forwarding (BEBA)* [42], es un proyecto dentro del marco de proyectos europeos *H2020*, que logró aumentar el rendimiento de *BOFUSS* en un orden de magnitud, pasando de una tasa de transferencia media de aproximadamente 185 Mbps a una capacidad que ronda los 1,19 Gbps.

- *ARP-Path OpenFlow Software Switch (ARP-Path OpenFlow Software Switch (AOSS))* [43], es un switch híbrido SDN donde se ha programado la lógica necesaria para ejecutar el protocolo ARP-Path [44]. Este protocolo permite al switch generar caminos de forma autónoma, es decir sin necesidad de consultar al controlador. Este desarrollo provoca que la cantidad de flujos que un controlador debe procesar sea mucho menor, pero a cambio produce que el controlador no conozca la distribución de tráfico de forma directa ya que no tiene forma de consultar las tablas de reenvío de ARP-Path. Este software switch es el dispositivo elegido para el desarrollo del mecanismo de recuperación híbrido propuesto dentro del marco de esta tesis.

- *OnLife* [45], es un desarrollo del proyecto CORD [46] de Telefónica, cuyo principal objetivo es acercar los servicios de red lo más cerca posible al usuario final para mejorar, de este modo, su calidad y crear una red limpia sin ningún protocolo heredado.

3.2.4.2 Open vSwitch

El segundo switch software SDN que se presenta en esta subsección es Open vSwitch (OvS), este dispositivo de red es un switch virtual multicapa bajo licencia Open Source Apache 2.0. OvS[47] está diseñado para conmutar paquetes a una velocidad muy superior a la de *BOFUSS* (incluso con la mejora introducida por el proyecto *BEBA* [42]), razón por la cual su uso en redes de altas prestaciones es bastante frecuente.

OvS, a diferencia del desarrollo llevado a cabo en *BOFUSS*, se ha implementado como dos procesos diferentes para aumentar su eficiencia y rendimiento. El primero de ellos se ejecuta en espacio de *Kernel*, mientras que el segundo proceso se ejecuta en espacio de *Usuario* (ver figura 3.20 [43]). La sección situada en espacio de *Kernel* tiene como función principal el tratamiento de los paquetes, almacenar las tablas de flujos y ejecutar las acciones definidas para los paquetes; mientras que la parte que funciona en espacio de *Usuario* se encarga de ejecutar las operaciones definidas por el protocolo OpenFlow, así como de la comunicación con el controlador.

Aunque OvS es un switch SDN de alto rendimiento sigue siendo compatible con protocolos estándar para su administración tales como NetFlow [48], sFlow [49], IPFIX [50], Remote Switch Port ANalyzer (RSPAN), Common Languaje Interface (CLI), Link Aggregation Control Protocol (LACP) o 802.1ag.

3.2.5 Controlador

Hasta este momento se han introducido los elementos de la *capa de infraestructura* como por ejemplo OvS y *BOFUSS*, dos implementaciones de switches compatibles con la arquitectura SDN, así como OpenFlow [35]), el protocolo de comunicación entre dicha capa y la *capa de control*. En esta subsección se va a presentar el elemento principal de la capa de *control*, el controlador.

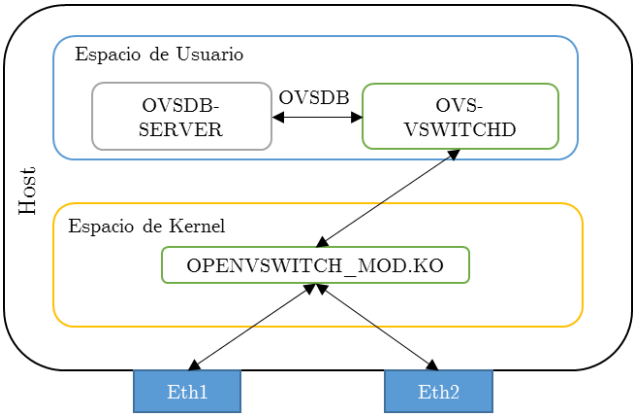


Figura 3.20: Estructura interna de OvS [43]

Este elemento se puede definir como un software intermedio o *middleware* situado entre las aplicaciones diseñadas por los gestores de la red y los switches que conforman físicamente la red. Este *middleware* permite al gestor abstraerse de la complejidad y dificultad de controlar los sistemas distribuidos que se pueden encontrar en las redes de comunicaciones tradicionales y realizar un control de una manera mucho más sencilla, cómoda y, en muchos casos, más efectiva.

El control de la red se establece, en la mayoría de los casos, mediante la instalación de reglas en los diferentes switches SDN. La instalación de las reglas por parte del controlador se puede realizar de tres formas diferentes: *Proactiva*, *Reactiva* y *Híbrida*. De forma *Proactiva*, es decir el controlador preinstala las reglas de todos aquellos flujos que conoce a priori. Este método es muy utilizado para disminuir el tiempo de establecimiento de los flujos en la red. En segundo lugar, se encuentra la instalación de reglas de manera *Reactiva*, en este caso el controlador instala en los switches las reglas según se las vayan solicitando, es decir, realiza una instalación de reglas salto a salto. Este método no es muy utilizado ya que aumenta mucho el tiempo de establecimiento de los flujos y, por lo tanto, introduce un retardo innecesario en la red. Por último, se encuentra el 'modo' *Híbrido*, donde el controlador instala las reglas bajo *demanda*, pero a diferencia del 'modo' *Reactivo*, también instala las reglas en todos los switches del camino desde la aparición del flujo en cuestión. Este método es el más utilizado ya que, aunque introduce un retardo debido a la instalación de las reglas, es menor que el retardo introducido por el 'modo' *Reactivo*. Además, permite una mayor flexibilidad que el 'modo' *Proactivo*, ya que, no obliga al controlador a tener que conocer a priori a todos los flujos que en algún momento puedan atravesar la red que él administra.

3.2.6 Tipos de controlador

Como se ha explicado, actualmente los controladores SDN son sistemas cuyo uso facilita la gestión de la red en gran medida. Esta razón ha llevado a muchas empresas a desarrollar nuevos controladores SDN, lo que ha producido un incremento en su número. En esta tesis se ha querido reflejar esa diversidad de controladores mediante la tabla resumen 3.7 [51], la cual expone una comparación entre los controladores más relevantes hasta el momento de la presentación de esta tesis.

3.2.7 Limitaciones de las arquitecturas SDN actuales

La arquitectura SDN ofrece ventajas evidentes que han permitido la expansión de este tipo de redes a redes de alto rendimiento, como las DCN. Algunas de estas ventajas son: poder controlar con una granularidad fina los distintos flujos que navegan por la red gracias a la posibilidad de modificar, sobre la marcha, las reglas de reenvío desde el controlador; la reconfiguración de varios switches en paralelo haciendo uso únicamente de

	Versión OpenFlow	Lenguaje	SSOO	Compatible OpenStack	Multiproceso	Open Source	API REST
NOX	v1.0	C++	Linux	No	Sí	Sí	No
POX	v1.0	Python	Linux, Mac Os and Windows	No	No	Sí	No
BEACON[52]	v1.0	Java	Linux	Sí	Sí	Sí	Sí
TREMA[53]	v1.3	Ruby and C	Linux	No	No	Sí	Sí
RYU	v1.0-v1.3	Python	Linux	Sí	No	Sí	Sí
ODL	v1.0-v1.3	Java	Linux	Sí	Sí	Sí	Sí
ONOS	v1.0-v1.3	Java	Linux	Sí	Sí	Sí	Sí

Tabla 3.7: Tabla de controladores [51]

las distintas API proporcionadas por la capa de control o la posibilidad de tener diferentes tipos de *hardware* (heterogéneos) dentro de una misma red, sin que ello introduzca una mayor complejidad en el control o gestión de la red. Asimismo, la comunidad científica ha utilizado la arquitectura SDN en muchas ocasiones para proponer avances dentro de las redes que en muchos casos como, por ejemplo, DevoFlow [54], Hedera [55] o Past: Scalable Ethernet For Data Centers (PAST) [56], se centran en las redes DCN.

Como es normal, este modelo arquitectónico SDN trae consigo una serie de inconvenientes como pueden ser: un coste de implantación muy elevado debido a la necesidad de desplegar todo un nuevo conjunto de switches compatibles con la arquitectura, la creación en muchos casos de una red paralela para el control de dichos elementos, la implantación de un controlador lo suficientemente potente como para ser capaz de atender todos los aspectos, sucesos y eventos que surgen en las redes, sin que ello pueda producir un cuello de botella, la escalabilidad de los mecanismos utilizados los controladores para obtener la información topológica de su red o la rigidez con la que *OpenFlow* define el *pipeline* de los dispositivos de red [57].

Respecto al problema de la creación de una red paralela para el tráfico de control, el propio protocolo *OpenFlow* define la posibilidad de utilizar la propia red de datos como red de control (modo *In-Band*). Sin embargo, muchos switches *OpenFlow* necesitan ser configurados manualmente para poder utilizar la red de datos como red de control, lo que aumenta los costes de instalación.

Referente a la necesidad de instalar un controlador extremadamente potente, la comunidad científica ha propuesto algunas soluciones. Por Kandoo [58] propone distribuir las tareas de control entre múltiples instancias del controlador, para así, disminuir la posibilidad de aparición de cuellos de botella en la Capa de Control. O propuestas que pretenden revertir parte de la lógica de control a los switches, para así reducir el uso del controlador y, por lo tanto, disminuir la necesidad de un *hardware* tan costos. Entre todas las propuestas de este tipo se desea citar: DevoFlow [54], OpenState [59] y Difane [60],

Sobre los mecanismos de adquisición de la información topológica, la comunidad científica ha propuesto una serie de protocolos, tales como, OFDP. Este protocolo es actualmente uno de los más utilizado por parte de los *Controladores* SDN para obtener la información topológica de la red subyacente. Este protocolo está basado en el protocolo

LLDP [61]. [62] demostró que OFDP tiene un serio problema de escalabilidad, ya que en redes de gran tamaño el número de paquetes generado es alto y puede sobrecargar al controlador.

Por último, con respecto a la falta de flexibilidad del *pipeline* definido por OpenFlow para los dispositivos compatibles con él, la comunidad científica ha propuesto diversas soluciones, de las que se desea destacar, Programming Protocol-independent Packet Processors (P4) [63], formulada por Open Networking Foundation (ONF). Este nuevo lenguaje para la programación de switches aporta muchísima más flexibilidad ya que permite una programación particularizada del *pipeline* de cada dispositivo.

3.3 Redes de datos con arquitecturas híbridas

Como se ha visto en los apartados anteriores existen dos arquitecturas muy diferentes, la arquitectura tradicional basada principalmente en sistemas distribuidos y la arquitectura SDN basada en una arquitectura centralizada. En esta sección se va a introducir una nueva arquitectura, la arquitectura híbrida. Esta arquitectura se puede definir como un conjunto de sistemas compuesto por entidades de diversos tipos y orígenes, para el caso de esta tesis esos sistemas serán los expuestos en las secciones 3.1 y 3.2.

Actualmente las redes híbridas formados por sistemas convencionales y sistemas SDN se pueden dividir en dos grandes grupos: las redes mixtas o híbridas [64], donde la unificación de ambos sistemas se establece a nivel global, es decir, se integran switches de ambas arquitecturas; y las redes formadas por switches híbridos, en este caso la unión se realiza a nivel de dispositivo. En este último caso, los dispositivos son capaces, por un lado, de trabajar como switches SDN (capaces de comunicarse con el controlador) y, por otro lado, de trabajar como switches tradicionales.

3.4 Migración de redes tradicionales a redes SDN

Se puede definir como red mixta o híbrida [65] toda red compuesta por elementos tradicionales, como, por ejemplo, switches, y elementos SDN tales como el switch software OvS, trabajando de manera conjunta. Por lo general, este tipo de redes surgen durante el proceso de migración necesario para transformar una red tradicional a una red definida por software. Debido a la complejidad asociada a los procesos de migración de las redes de gran tamaño, la comunidad científica ha realizado una serie de estudios con el objetivo de crear un conjunto de reglas o pautas capaces de mitigar los problemas producidos en las comunicaciones durante dicha migración.

Rouse [65] establece que para una correcta migración de las topologías jerárquicas híbridas utilizadas en las DCN, se deben seguir las siguientes pautas:

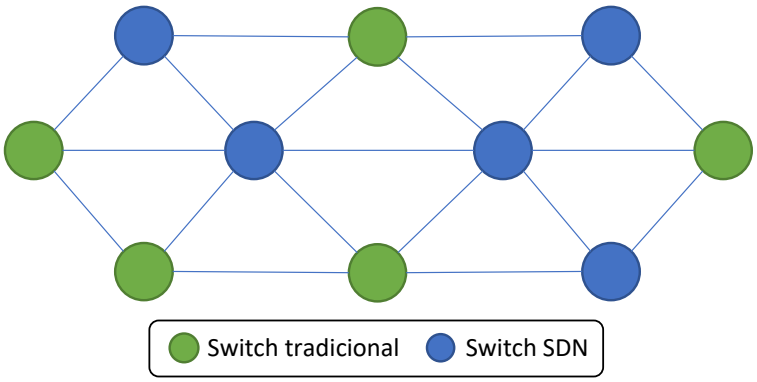


Figura 3.21: Ejemplo de una red mixta

- Los switches situados en la capa superior, denominada capa *CORE*, deben ser compatibles con los protocolos IP y Multiprotocol Label Switching (MPLS) y recomienda el uso de *enrutadores*, debido a que su ubicación dentro de la red puede conllevar que una de sus funciones principales sea la de pasarela con redes externas.
- Para los elementos establecidos en la capa intermedia, denominada capa *Agregada*, propone el uso de switches tradicionales ya que su principal función es la de transmitir los paquetes entre la capa *CORE* y *Top of Rack (ToR)*.
- Por el contrario para los switches de acceso a la red, situados en la capa inferior, denominada capa *ToR*, propone el uso de elementos SDN debido a que al situarse en la capa de entrada de los flujos permite al controlador establecer cada una de las rutas de una manera precisa.

Por el contrario si la topología de la red no es jerárquica, existen algunas soluciones propuestas por la comunidad como por ejemplo, la propuesta por [66], la cual propone un mecanismo basado en sistemas heurísticos para establecer la mejor posición de los switches SDN y así establecer un red híbrida como la representada por la figura 3.21. Sin embargo, actualmente siguen surgiendo problemas de compatibilidad entre elementos SDN y tradicionales.

3.4.1 Control en redes mixtas o redes híbridas

Como se ha expuesto en la sección anterior, las redes híbridas surgen principalmente en el proceso de migración entre una red tradicional y una red SDN debido al alto coste asociado a la migración de todos los equipos de forma simultánea. El principal problema que se genera al crear este tipo de redes es la gestión y control, debido a que se trata de dos sistemas diferentes, uno tradicionalmente compuesto por protocolos de control distribuidos y el otro con una gestión principalmente centralizada. La solución a este problema requiere no solo de personal altamente cualificado sino también de la utilización de diferentes soluciones complejas y, en algunos casos, con un alto coste asociado.

La comunidad científica ha propuesto algunas soluciones como las citadas a continuación:

- Soluciones basadas en un sistema capaz de comunicarse tanto con switches tradicionales como con switches SDN. [67] ofrece una herramienta de control centralizada capaz de establecer comunicación tanto con los switches tradicionales como con los switches SDN. Asimismo, también tiene la capacidad de establecer caminos para garantizar el correcto funcionamiento de la red. Y por otro lado, [68], ofrece una herramienta centralizada capaz de comunicarse con diferentes tipos de switches aprovechando el conocimiento de sus interfaces de comunicación. Por ejemplo, puede comunicarse con switches SDN mediante *OpenFlow*, y con switches *tradicionales* mediante Simple Network Management Protocol (SNMP), WebService o CLI. Esta última propuesta establece que el encaminamiento de los flujos dentro de las redes híbridas se debe realizar mediante la creación de circuitos virtuales basados en Virtual Local Area Networks (VLANs).

- Soluciones donde cohabitan diferentes entidades capaces de comunicarse únicamente con un tipo de dispositivos, es decir, solo pueden comunicarse con dispositivos tradicionales o con dispositivos SDN, como la propuesta en [69]. Esta propuesta pretende configurar de forma automática a los switches SDN según vayan siendo introducidos en una red tradicional. Para ello establece, primero, que el nuevo switch SDN debe comunicarse con un servidor Dynamic Host Configuration Protocol (DHCP) para obtener una dirección IP con la que poder comunicarse con el controlador SDN, y segundo, que un módulo especialmente diseñado reconfigure los switches tradicionales para permitir la comunicación entre el nuevo switch SDN con el controlador.

- Soluciones como Panopticon [70] que hace uso de un controlador estándar SDN para controlar los dispositivos SDN mediante mensajes *OpenFlow* y paquetes ARP para controlar los dispositivos tradicionales aprovechando el proceso de aprendizaje de que disponen.

3.4.2 Redes formadas por switches híbridos

Otra manera de crear redes híbridas es mediante la incorporación de elementos híbridos, es decir, switches capaces de trabajar utilizando sistemas tradicionales y/o SDN. Este tipo de dispositivos ya están contemplados en el estándar de *OpenFlow* en su versión 1.3 [35]. En dicha versión del protocolo se define que un switch OpenFlow puede conocer y/o ejecutar protocolos y/o procesos de conmutación tradicionales sin que ello pueda afectar ni a la red ni al dispositivo.

Existen múltiples fabricantes que ya tienen dentro de su catálogo dispositivos *híbridos* SDN/tradicionales. Estos switches siguen dos diseños diferentes principalmente: el primer

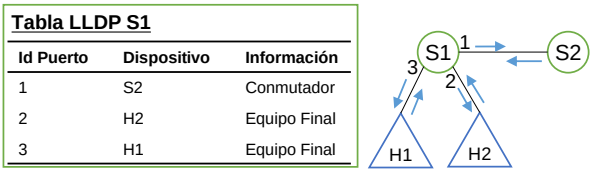


Figura 3.22: Ejemplo de funcionamiento del protocolo LLDP

gran bloque se caracteriza por introducir dentro de la lógica del switch la capacidad de comportarse siguiendo la lógica de un switch *tradicional* o la de un switch SDN, de tal forma que el usuario puede elegir el modo de funcionamiento del dispositivo, pero no puede elegir que el dispositivo funcione utilizando los dos modos a la vez; mientras que, el segundo gran bloque también inserta los dos modos de funcionamiento, pero esta vez el usuario elige qué puertos se van a comportar de una manera y qué puertos de otra, es decir, en este caso el dispositivo ejecuta en paralelo los dos modos, pero cada uno de los puertos solo puede estar asociado a uno de ellos.

Por otro lado, la ONF ha propuesto dividir el plano de datos en subconjuntos separados y disjuntos donde cada uno de esos subconjuntos se controle a través de diferentes estancias del *Plano de Control* [71]. El problema de este enfoque radica principalmente en dos puntos, el primero es el lugar donde realizar esa separación en subconjuntos y el segundo es la creación de funciones de control duplicadas debido a esa separación. Para solucionar esta duplicidad de funciones propone mover las funciones de transporte directamente a los switches y las funciones relacionadas con los servicios unificarlas en controlador, de esta manera se evita la duplicidad de funciones.

3.5 Protocolos de descubrimiento de topologías

Uno de los puntos fundamentales en el paradigma de las redes SDN, es el conocimiento de la topología por parte del controlador, es por este motivo que se haya decidido hacer un pequeño estudio de algunos de los protocolos de descubrimiento de topologías más relevantes en el momento de presentación de esta tesis.

3.5.1 LLDP: Link Layer Discovery Protocol

LLDP [61] es un protocolo de capa dos perteneciente al Institute of Electrical and Electronics Engineers (IEEE) (IEEE 802.1AB) cuya finalidad es el descubrimiento de los vecinos conectados directamente a un dispositivo de la red. Su funcionamiento es muy sencillo (ver figura 3.22), se basa en que cada uno de los dispositivos que utilizan LLDP intercambian su información con cada uno de sus vecinos. Dicha información se almacena en la memoria interna del dispositivo. Para que LLDP pueda lograr su objetivo necesita enviar una trama especial (*LLDP Frame*) a través de cada una de sus interfaces

Chassis ID	Port ID	Time To Live	Optional TLV	End LLDPU

Figura 3.23: Trama LLDP

Port Description	System name	System Description	Port capabilities	Management address

Figura 3.24: TLV opcionales en la trama LLDP

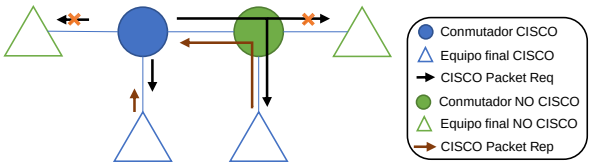


Figura 3.25: Ejemplo de funcionamiento protocolo CDP

Ethernet. Dicha trama está compuesta obligatoriamente por (ver figura 3.23): identificador del dispositivo (*Chassis ID*), identificador del puerto (*Port ID*), tiempo de vida del paquete (*Time-To-Live*) y del indicador de final de trama LLDP. La trama opcionalmente puede contener otra información adicional tal como: la descripción del puerto, el nombre, descripción y funciones del dispositivo de red, además de una dirección de gestión del switch (ver figura 3.24).

En resumen LLDP no es un protocolo de descubrimiento de topologías al uso, sino un protocolo de descubrimiento de vecinos. No obstante, a partir de la información de los vecinos se puede obtener la información topológica como se verá más adelante.

3.5.2 CDP: Cisco Discovery Protocol

Otro protocolo de descubrimiento de vecinos, pero en este caso de licencia privada, es Cisco Discovery Protocol (CDP) [72]. Este protocolo tiene un funcionamiento muy similar al anterior ya que al igual que LLDP funciona mediante intercambio de mensajes y su objetivo es el mismo, descubrir y almacenar la información de los diferentes vecinos de un nodo.

Para ver más clara la diferencia de funcionamiento se muestra la figura 3.25, en la cual se puede observar como si se usa CDP como protocolo de descubrimiento de vecinos, todos los elementos finales que no sean Cisco descartan el paquete mientras que los switches de la red de otra plataforma diferente a Cisco simplemente los propagan. De esta manera el descubrimiento de los vecinos no es totalmente completa. Mientras que usando LLDP, cada uno de los elementos intercambia mensajes con todos sus vecinos para de esta manera obtener una información completa.

3.5.3 OFDP: OpenFlow Discovery Protocol

Tras el aumento del número de redes que utilizan las nuevas redes SDN, la comunidad científica trabajó en un mecanismo capaz de descubrir de forma más o menos eficiente la información sobre la topología de la red. Tras algunos estudios se vio que la forma más sencilla de obtener dicha información era adaptar el protocolo LLDP para este nuevo paradigma, surgiendo así OFDP [10]. OFDP proporciona al controlador la información necesaria para generar el grafo de la topología en el controlador haciendo uso de los mismos mensajes definidos en el protocolo LLDP (ver figuras 3.23 y 3.24) y de un postprocesado de datos en el controlador.

El funcionamiento de este protocolo se muestra en la figura 3.26. El protocolo se inicia en el controlador, el cual envía a cada switch SDN tantos mensajes LLDP encapsulados en paquetes *Packet_Out* como puertos tenga dicho switch. Cada vez que un paquete *Packet_Out* es recibido por switch SDN, será desencapsulado por el switch para obtener la trama LLDP original, para posteriormente, reenviar dicha trama al vecino a través del puerto indicado por el controlador en el paquete *Packet_Out*. Por último, el switch SDN vecino que recibe el paquete LLDP, encapsula dicho paquete en *Packet_In* y lo envía al controlador para ser procesado. El controlador, cada vez que recibe un *Packet_In*, obtiene los identificadores de los switches SDN unidos a través de un enlace, así como los puertos de salida y entrada de dicho enlace. Por lo tanto, una vez el controlador recibe todos los paquetes LLDP enviados a la red podrá recrear el grafo que representa la topología de la misma.

3.5.3.1 OFDPv2: OpenFlow Discovery Protocol v2

Como es de suponer, el número de mensajes utilizados por OFDP aumenta si incrementa el número de puertos que existen en la red, lo que puede producir un problema de escalabilidad. Por ello, surgió OpenFlow Discovery Protocol Version 2 (OFDPv2) [73], el cual establece que no hace falta enviar un paquete *Packet_Out* por cada puerto que existe en la red, sino que con mandar un *Packet_Out* por cada switch SDN es suficiente. La mejora radica en que el switch SDN debe difundir el mensaje recibido en vez de reenviarlo por un único puerto consiguiendo una disminución apreciable del número de mensajes.

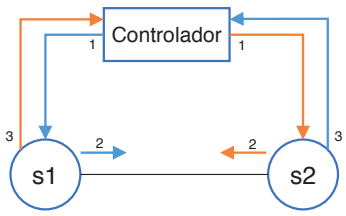


Figura 3.26: Ejemplo de funcionamiento protocolo OFDP

3.5.4 sOFTDP: Secure and Efficient Topology Discovery Protocol for SDN

Con la aparición de OFDP, la comunidad pasó a disponer de un servicio fiable para descubrir la topología, o eso se pensaba en un principio, pero la realidad es que OFDP tiene grandes problemas como son: falta de seguridad, baja escalabilidad o ineficiencia, todos ellos estudiados por Abdelhadi Azzouni en [74].

Abdelhadi Azzouni en [74] propone Secure and Efficient Topology Discovery Protocol for SDN (sOFTDP) como posible solución a los problemas de seguridad de OFDP. Esta solución establece diferentes modos de actuar dependiendo de la situación:

- En el caso de tener que refrescar la información topológica, propone seguir el mismo proceso que el establecido en OFDPv2 pero sustituyendo las MACs de los equipos por un valor calculado mediante la aplicación de una función *HASH* a la MAC de dichos equipos. De esta manera pretende evitar ataques mediante el uso de técnicas de *spoofing*.
- Para el caso de un alta nueva de un switch SDN en la red, propone que el controlador, una vez haya recibido los paquetes de aviso de cambio de estado en los puertos de los switches vecinos a través de los paquetes *Port_Status* correspondientes, debe enviar paquetes LLDP únicamente a las interfaces que hayan notificado dicho cambio, y solo actualizar la información del grafo topológico una vez hayan sido recibidos de vuelta todos y cada uno de los paquetes LLDP enviados.
- En cuanto al caso de una nueva alta de un enlace entre dos switches SDN, se propone que el controlador envíe los mensajes LLDP únicamente a las interfaces afectadas por el nuevo enlace.
- Finalmente, para el caso de caída de un switch SDN o un enlace de la red, sOFTDP propone que el controlador no envíe ningún mensaje LLDP sino que use la información proporcionada por los mensajes *Port_Status* recibidos desde los switches afectados.

3.5.5 SD-TDP: Software-Defined - Topology Discovery Protocol

Para mejorar la eficiencia de los servicios de descubrimiento de topologías Leonardo Ochoa-Aday propuso Software-Defined Topology Discovery Protocol (SD-TDP) [75, 76]. Este protocolo se compone de tres fases distintas. La primera fase comienza con él envío por parte del controlador de tantos mensajes *Packet_Out* como interfaces activas tengan los switches SDN conectados a él. Estos *Packet_Out* encapsulan un mensaje propio del protocolo denominado *TDP-Request*. Una vez el switch recibe el paquete cambia su estadio inicial denominado como *Standby* a uno de los dos estados definidos dentro de su diagrama de estados: *Padre* o *Activo*. En la segunda fase SD-TDP establece que, por un lado, los switches que se encuentran en el estado denominado como *Padre*, a partir de ahora

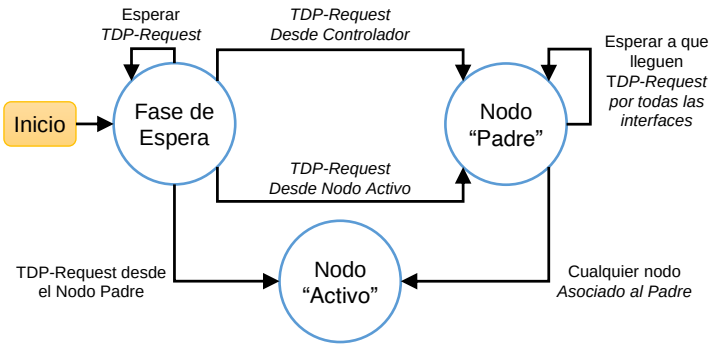


Figura 3.27: Máquina de estados de SD-TDP

switches *Padres*, son los encargados de reunir la información procedente de los switches asociados a cada uno de ellos y de enviar dicha información al controlador. Por otro lado, que los switches que se encuentran en el estado denominado como *Activo*, a partir de ahora switches *Activos*, deben, después de recibir el mensaje *TDP-Request*, responder con su información y reenviar el mensaje *TDP-Request* recibido por el resto de las interfaces disponibles. El diagrama del proceso completo se muestra en la figura 3.27.

3.5.6 ETDP: Enhanced Topology Discovery Protocol

Enhanced Topology Discovery Protocol (ETDP) [77] es un protocolo de descubrimiento de topologías que surgió con posterioridad a la publicación del protocolo de descubrimiento topológico propuesto en esta tesis. Este protocolo propone crear un árbol de control con raíz en el controlador mediante la propagación de mensajes *topoRequest*.

ETDP define a su vez tres tipos distintos de roles para los switches de la red:

- *Leaf*: todos aquellos switches que disponen de un único vecino.
- *V-leaf*: todos aquellos switches que disponiendo de varias interfaces solo una de ella se puede conectar con el controlador.
- *Cores*: el resto de los switches de la red.

También define varios estados posibles para las interfaces de los switches:

- *Standby*: es el estado correspondiente a una interfaz activa pero que no pertenece al árbol de control.
- *Parent*: es el estado que se le otorga al puerto por el que el switch recibió el primer mensaje *topoRequest*, razón por la cual cada switch solo puede tener un único puerto de este tipo. Este puerto, además de forma parte del árbol de control, es un puerto que permite enviar los paquetes al nivel inmediatamente *superior* dentro de la jerarquía del árbol, es decir, los paquetes se acercan de la raíz del árbol.

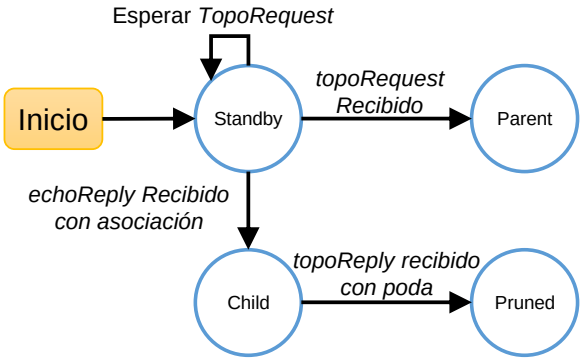


Figura 3.28: Máquina de estados de las interfaces en ETDP

- *Child*: es el estado que adquiere un puerto cuando por él se ha recibido un mensaje *echoReply*. Este puerto, además de forma parte del árbol de control, es un puerto que permite enviar los paquetes al nivel inmediatamente *inferior* dentro de la jerarquía del árbol, es decir, los paquetes se alejan del nodo raíz del árbol.

- *Pruned*: es el estado que ostenta un puerto *Child* tras recibir un mensaje *topoReply*. Este hecho, además, significa que está directamente conectado con un switch *Leaf* o *V-Leaf*.

Una vez conocidos los distintos tipos de switches y los distintos posibles estados de sus interfaces se presenta la figura 3.28, la cual muestra un diagrama de estados que representa los cambios entre los diferentes estados asociados a las interfaces.

El mecanismo propuesto por ETDP necesita iniciarse en un punto donde todos los switches se encuentran en un estado de *no descubrimiento*. ETDP establece su punto de partida en el instante en que el controlador envía los mensajes *topoRequest* encapsulados en distintos paquetes *Packet_Out*, se puede decir que aquí comienza una fase de exploración de la red mediante difusión.

La 'segunda fase' comienza cuando, una vez iniciada la difusión de las tramas *topoRequest* por parte de los switches receptores de los paquetes *Packet_Out*, comienzan a recibirse cada una de las copias de los mensajes *topoRequest*. En esta fase, los switches realizan dos acciones: en la primera, vuelven a difundir la trama *topoRequest* por todas sus interfaces menos por la de entrada, mientras que, en la segunda, contestan dicho mensaje con un mensaje *topoReply*. Esta respuesta habilita el intercambio de información entre el switch emisor del paquete *topoRequest* y el switch receptor de dicho paquete, además, también permite establecer una medida del RTT entre dichos dispositivos.

Por último ETDP establece que toda la información intercambiada por los diferentes switches de la red se debe almacenar en switches catalogados como *Cores*. Estos switches de forma periódica envían dicha información al controlador para, mediante un procesado interno, obtener la topología de la red.

3.6 Protocolos de árbol de expansión

En esta nueva sección se van a estudiar alguno de los protocolos más característicos que hacen uso de uno o varios árboles de expansión para el encaminamiento del tráfico.

3.6.1 STP: Spanning Tree Protocol

STP [2] es un protocolo que se encuentra desarrollado dentro del estándar IEEE 802.1D. STP opera en la capa de enlace y busca, mediante la prohibición del uso de ciertos enlaces en la red, evitar la aparición de bucles. STP para llevar a cabo su objetivo se apoya en algoritmos basados en la creación de árboles de expansión. STP, primero establece qué dispositivo va a actuar como el switch raíz (*Root Bridge*) del árbol, para ello, y por norma general, elige el switch cuya MAC tenga un menor valor.

Posteriormente, establece el rol de las interfaces asociadas a cada uno de los switches que conforman el árbol de expansión. Los distintos roles definidos dentro del protocolo son los siguientes:

- *Root port*, es la interfaz física con un menor coste para llegar al switch raíz. Al igual que cuando el sistema debe elegir a su *switch raíz*, si se diera el caso de que dos interfaces tuvieran el mismo coste en el camino que les une con el switch raíz se define como *Root port* aquel puerto cuya MAC comience por los dígitos de menor valor.
- *Designated port*, es el rol asignado a aquellas interfaces que tienen habilitado el reenvío de tráfico. Estas interfaces tienen como principal objetivo la propagación del árbol de expansión hacia otros switches de la red.
- *Backup port*, es el rol definido para aquellas interfaces a las que se les ha prohibido el reenvío de tráfico, el objetivo de este rol es evitar la aparición de bucles dentro de la red.
- *Disabled port*, es el rol establecido para todos aquellas interfaces que se encuentran desactivados.

Por último, STP establece el estado de dichas interfaces entre los siguientes:

- *Desactivado o 'Disabled'*, son los puertos deshabilitados manualmente o por un error. Los puertos cuyo estado sea *desactivado* no toman parte dentro del progreso del protocolo.
- *Bloqueo o 'Blocking'*, son los puertos que han obtenido el rol de *no designados*. Estos puertos tienen deshabilitado la capacidad de reenviar tramas para evitar la aparición de bucles.

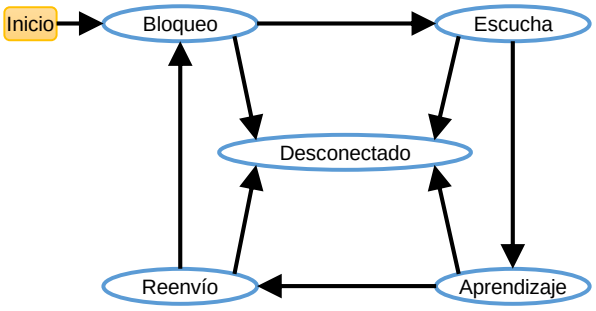


Figura 3.29: Máquina de estados de los puertos en STP

- *Escucha o 'Listening'*, este estado permite que un puerto pueda ser seleccionado como *puerto raíz* o *puerto designado*. En este estado el puerto acepta recibir y enviar Bridge Protocol Data Units (BPDU) pero no tramas.

- *Aprendizaje o 'Learning'*, es un estado similar al estado anterior, pero a diferencia de su predecesor, este estado si permite al puerto aprender nuevas direcciones MAC recibidas en las BPDUs.

- *reenvío o 'Forwarding'*, en este estado el puerto es capaz de enviar y recibir todo tipo de tramas, incluyendo las BPDUs del protocolo, del mismo modo, también permite puede aprender nuevas direcciones MACs.

El diagrama de cambio de estados en los puertos se presenta en la figura 3.29.

El modo de funcionamiento de STP puede dividirse en tres grandes pasos: elección del *switch raíz*, elección del *puerto raíz* y elección de los *puertos designados*.

El primero paso es elegir el *switch raíz* del árbol, para ello los switches intercambian tramas (*BPDU*) en los que cada switch ha introducido previamente su identificador. A medida que dichas tramas van llegando a cada uno de los switches, estos van actualizando su información, para ello comparan el valor del identificador que han recibido con el suyo propio. Fijando como *switch raíz* el switch cuyo identificador sea el de menor valor.

El segundo paso es identificar la posición del resto de switches en relación con el *switch raíz*, para ello cada uno switch debe elegir su *puerto raíz*. Este proceso se realiza mediante el cálculo, por parte de cada switch, del coste que tiene la ruta desde cada uno de sus puertos al *switch raíz*. Finalmente, el switch designa como *puerto raíz* aquel puerto cuyo coste sea menor.

El último paso es identificar los *puertos designados* de cada uno de los switches. Cada switch debe seleccionar un *puerto designado* para cada segmento de la red, en este contexto se puede entender segmento como enlace entre dos switches. Para determinar que puertos son los *designados*, cada switch compara el coste asociado a cada puerto con el coste de la ruta que recibe a través de la trama BPDU entrante, si el coste de su puerto es menor lo establece como *designado* en caso contrario como *no designado*.

3.6.2 RSTP: Rapid Spanning Tree Protocol

Rapid Spanning Tree Procotol (RSTP) [78] es un protocolo estándar del IEEE especificado en IEEE 802.1w, RSTP pertenece al conjunto de protocolos que funcionan mediante árbol de expansión dentro de la capa de enlace. Este protocolo se puede ver como evolución/mejora del protocolo STP.

El desarrollo de RSTP impuso como uno de sus objetivos la reducción de forma significativa del tiempo que necesitaba STP para converger. Para ello, RSTP propone algunos cambios muy radicales respecto a STP. El primero, propone una reducción del número de estados que existen para las interfaces, pasando de los cinco estados establecido por STP a tres: Descarte, Aprendizaje y Reenvío, es decir elimina el estado de escucha. El segundo cambio consiste en una ligera modificación de las tramas utilizadas para el intercambio de información, pero manteniendo la compatibilidad de estas nuevas BPDU con las antiguas para que los dispositivos que hicieran uso del protocolo STP pudieran ser compatibles con el nuevo protocolo. Las nuevas tramas, denominadas Bridge Protocol Data Units Version 2 (BPDUv2), de RSTP incluyen el rol del puerto que origina la trama dentro del campo *Port Role*. El tercer cambio introducido por RSTP, es que, mientras STP solo utiliza dos bits (*Topology Change* y *Topology Change Acknowledgment*) de los siete posibles del campo de *flags*, este nuevo protocolo hace uso de todos ellos (ver figura 3.30), para, por un lado, indicar el estado y el rol del puerto que ha enviado la trama y, por otro lado, indicar la propuesta o la aceptación ante una modificación en la red. La cuarta modificación establece que todos los elementos de la red puedan generar tramas BPDUv2 y además han de hacerlo de forma periódica, mientras que STP solo permite generar tramas BPDU al root. El quinto, y más importante, es la eliminación por parte de RSTP de la mayoría de los temporizadores (*timers*), pasando de un modelo basado en plazos a un modelo basado en eventos. Este último cambio es el que realmente permite una reducción del tiempo de convergencia ya que a diferencia de STP, RSTP no debe esperar a un tiempo determinado para realizar una acción sino que, la generación de un evento (como la recepción de una trama) va a conllevar la realización de una acción.

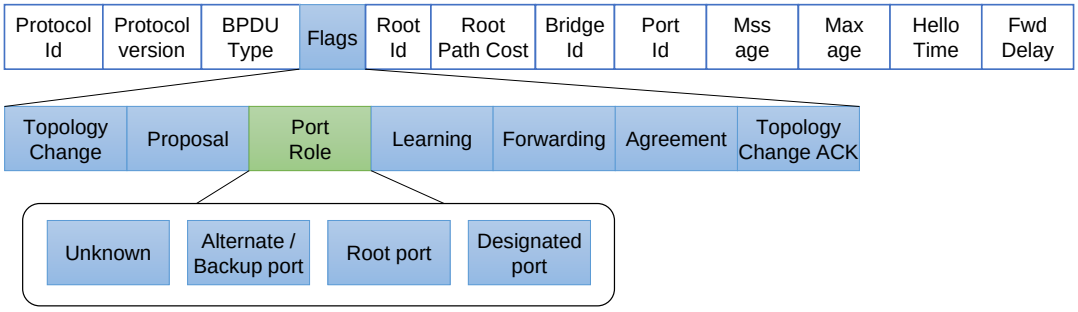


Figura 3.30: Flags de trama RSTP

Estas modificaciones traen consigo algunas mejoras en el rendimiento de RSTP. Al no depender de *timers*, RSTP necesita menos tiempo para converger que su predecesor (STP). También, gracias a los nuevos mecanismos que implementa, permite cambiar el rol de los puertos (*puerto raíz* o *puerto designado*) sin tener que pasar por el estado de *reenvío*, lo que permite reducir el tiempo de convergencia en el caso de haber cambios en la red. Pero, aunque consigue un mejor tiempo de convergencia, sigue teniendo el mismo problema que STP en cuanto se habla de balanceo de carga, ya que al igual que STP sigue sobrecargando los enlaces que pertenecen al árbol.

3.6.3 MSTP: Multi Spanning Tree Protocol

Multi Spanning Tree Protocol (MSTP) [79], se puede definir como otro intento de mejorar el protocolo estándar STP. En esta ocasión el objetivo era mejorar el balanceo de la carga de STP. Como es sabido, STP establece un único árbol de expansión en la red, lo que provoca que todo el tráfico sea encaminado por dicho árbol y, por lo tanto, se produce una sobrecarga de tráfico en dichos enlaces ya que deshabilita el resto de los enlaces. Por ello, MSTP propone la creación de múltiples árboles de expansión dentro de la red e introduce algunos nuevos conceptos para conseguir este objetivo.

- *Región*, este concepto representa a cada grupo de switches que comparten el mismo identificador de región y la misma VLAN asignada. Se usan principalmente para aumentar la escalabilidad del protocolo y disminuir el tiempo de reacción ante cualquier modificación en la red.

- *Instancias*, este concepto hace referencia al número de árboles vivos que coexisten en una misma red. Como es bien sabido, el número de árboles que puede coexistir al hacer uso de STP o RSTP es uno, por lo tanto, el número de instancias es uno. En el caso de MSTP el número máximo de instancias es de 64 en cada región. Se usan principalmente para optimizar el uso de la red.

- *Árbol de expansión común*, es el árbol que administra la conexión entre las diferentes regiones.

Para ello, MSTP aprovecha que suelen coexistir diferentes VLAN dentro de la red y define un árbol de expansión para cada una de ellas, es decir, la idea general es aplicar el protocolo STP para cada una de las VLAN existentes en la red, de tal manera, que cada una de ellas tenga su propio árbol de expansión con parámetros de configuración y switch raíz distinto y, por lo tanto, establecer un mejor balanceo de la carga mediante el uso de un mayor número de enlaces pues aquellos que deshabilita un árbol STP para una VLAN pueden pertenecer al árbol de otra VLAN y así utilizarse también.

3.7 Protocolos de establecimiento de caminos en capa de enlace

A continuación se va a proceder a presentar alguno de los protocolos más relevantes para el encaminamiento de caminos dentro de la capa de Enlace, es decir haciendo uso de las direcciones de enlace (MACs) y no mediante las direcciones de red (IPs).

3.7.1 SPB: Shortest Path Bridging

Continuando con la misma filosofía de creación de múltiples árboles dentro de una red propuesta por MSTP surgió el proyecto Shortest Path Bridging (SPB) [80], el cual se convirtió en estándar (*IEEE 802.1aq*) en el año 2012. Este protocolo propone el encaminamiento mediante *Shortest Path* o caminos mínimos. Para ello, SPB propone establecer múltiples árboles en la red donde cada uno de esos árboles tenga como raíz un switch distinto de la misma, de esta manera, SPB dispone de tantos árboles como switches de la red para el encaminamiento de los flujos. Los árboles creados por SPB son compatibles con MSTP y VLAN lo que facilita mucho su implantación en determinados entornos. Una de las principales características de SPB es en la manera que tiene de encaminar el tráfico, ya que pasa de utilizar un enrutamiento basado en el protocolo de estado de enlaces Intermediate System-Intermediate System (IS-IS).

SPB dispone de dos modos de funcionamiento diferentes: *Shortest Path Bridging-MAC* y *Shortest Path Bridging-VID*. *Shortest Path Bridging-MAC* se utiliza en todas aquellas redes donde es, por seguridad, necesario ocultar las direcciones MAC de los equipos finales. Para ello, este modo, hace uso de un encapsulado *MAC-in-MAC* de las tramas en los nodos frontera. Mientras que, *Shortest Path Bridging-VID* se utiliza principalmente en las redes donde no es fundamental ocultar las direcciones físicas. Este modo reutiliza la cabecera *VLAN* para dar compatibilidad con otros protocolos basados en STP.

3.7.2 TRILL: Transparent Interconnection of Lots of Links

Transparent Interconnection of Lots of Links (TRILL) [81] es un protocolo perteneciente a la Internet Engineering Task Force (IETF). TRILL se sitúa entre las capas dos y tres del modelo OSI, lo que le permite hacer uso de funcionalidades de ambas capas. Está basado en el protocolo de estado de enlaces IS-IS, lo cual le permite enrutar tráfico de datos (característica propia del nivel de red) y, además, permite la movilidad de los equipos finales (característica propia del nivel de enlace).

TRILL está especialmente diseñado para su implantación en *redes campus*. TRILL define a los switches que conforman sus redes como switches Routing Bridge (RBridge). Esta denominación solo indica que el dispositivo tiene implementado la lógica de TRILL dentro de su firmware.

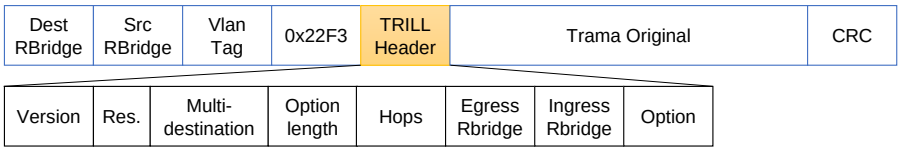


Figura 3.31: Trama TRILL

Respecto al enrutamiento de paquetes, TRILL lo realiza haciendo uso de la información proporcionada por el protocolo *IS-IS*, es decir, cada uno de los switches RBridge confecciona su tabla de enrutamiento a partir de la información que le proporciona *IS-IS*, un protocolo de estado del enlaces característico de la capa de red.

Además, TRILL también define una nueva cabecera para todas las tramas que recorran su red. TRILL hace uso de un proceso de encapsulado *MAC-in-MAC* para incorporar dicha cabecera al cada paquete sin modificar su contenido. La trama resultante tras el proceso de encapsulado llevado a cabo por TRILL se muestra en la figura 3.31. Esta nueva cabecera es actualizada por cada uno de los RBridge y eliminada en el RBridge frontera destino. Por tanto, la primera vez que la trama entra en dominio *TRILL*, el switch RBridge frontera encapsula dicha trama para obtener una trama como la representada en la figura 3.31. El RBridge frontera completa el valor de los campos: dirección del siguiente RBridge (*Dest RBridge*), dirección del RBridge actual (*Src RBridge*), identificador de 16 bits del RBridge de salida del dominio TRILL (*Egress RBridge Nickname*) identificador 16 bits del RBridge de entrada al dominio TRILL (*Ingress RBridge Nickname*) pertenecientes a la cabecera TRILL. Por su parte, cada uno de los RBridge que reciben la trama deben calcular el siguiente salto del camino que va a seguir la trama y actualizar la información de los campos *Dest RBridge* y *Src RBridge*, y por último, el switch frontera RBridge destino desencapsula el paquete TRILL para obtener y reenviar la trama original.

3.7.3 ECMP: Equal Cost MultiPath

ECMP[3] es el protocolo más utilizado y, por ello, considerado el estándar *de facto* para el encaminamiento del tráfico en redes con múltiples caminos entre un origen y un destino. ECMP surgió para el encaminamiento en capa de red (capa tres) pero actualmente existen implementaciones para su uso en capa de enlaces (capa dos) y, en muchos casos, compatibles con protocolos como TRILL y SPB. Tiene como objetivo el establecimiento de caminos entre un par de switches, eligiendo uno de entre todos los posibles caminos existentes entre ellos. Para realizar esta elección, ECMP hace uso de otro protocolo capaz de obtener dichos caminos como pudiera ser, el protocolo OSPF.

ECMP para fijar el camino, necesita en primer lugar, definir cuál va a ser el parámetro de medida que permita establecer el peso de cada uno de los diferentes caminos a los protocolos o algoritmos subyacentes (OSPF o *Dikjtra*). Este parámetro, suele ser el número de saltos entre el origen y el destino, mientras que en otros podría ser los pesos asignados

manualmente o el retardo del camino. En segundo lugar, el mecanismo de selección del camino entre todos los posibles de igual coste, este mecanismo suele ser mediante elección aleatoria, pero también puede definirse bajo cualquier otro criterio.

ECMP dispone de dos modos de funcionamiento principalmente, el primer modo es un reparto por paquete, es decir cada paquete se manda por uno de los caminos de igual coste disponibles. Este reparto puede lograr un mejor balanceo de la carga, ya que, utiliza todos los caminos disponibles de una manera equitativa, pero puede provocar una sobrecarga de procesamiento en los equipos finales debido a que los paquetes pueden llegar desordenados. El segundo modo, define un reparto por flujo, es decir, todos los paquetes que pertenezcan a un mismo flujo toman el mismo camino. Para realizar este encaminamiento ECMP tiene en cuenta varios parámetros pertenecientes a las cabeceras del paquete, tales como: la dirección MAC origen, la dirección MAC destino, el puerto TCP origen o el puerto TCP destino. Este último modo, a diferencia del anterior, necesita guardar estado en los switches de la red, pero a cambio, garantiza que los paquetes lleguen ordenados y, por lo tanto, aunque aumenta el procesamiento en los switches de la red lo disminuye en los equipos finales.

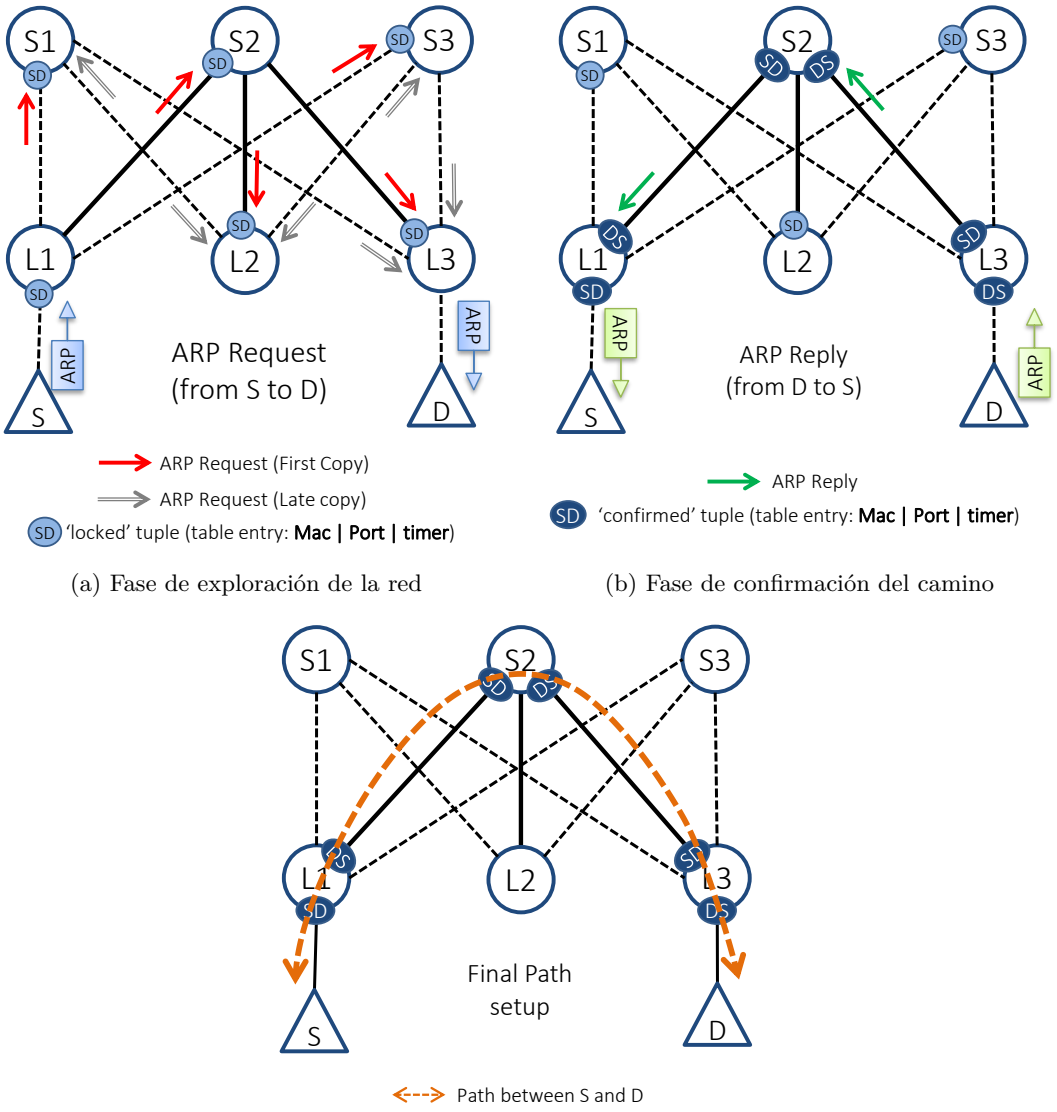
3.7.4 ARP-Path protocol

El protocolo *ARP-Path* [44] define un mecanismo para la generación de caminos en redes de capa dos. Este protocolo hace uso de la necesidad que tienen los equipos finales de conocer las direcciones MAC de todos los destinos de sus flujos, para ello una vez enviado el paquete de difusión *ARP-Request* por parte de uno de los equipos finales de la red, los switches guardan en su tabla la relación entre el puerto de entrada de dicho paquete, la MAC origen del mismo y el instante en que recibió dicho paquete, como se observa en la figura 3.32a.

ARP-Path tiene un mecanismo para la prevención de bucles muy sencillo y eficaz, para llevarlo a cabo comprueba, a la llegada de un paquete de difusión, si la MAC origen del paquete se encuentra en su tabla, en caso afirmativo comprueba dos cosas: la primera, que el puerto de entrada de dicho paquete no coincida con el puerto de entrada almacenado en la tabla y, la segunda, que el *timer* asociado a dicho registro esté en activo. Si las dos condiciones se dan el sistema descarta el paquete, en caso contrario reenvía el paquete por todos los puertos menos por el de entrada y actualiza el registro de su tabla.

Una vez el equipo final destino del paquete *ARP-Request* contesta con el paquete unicast *ARP-Reply* cada uno de los switches que recibe este paquete introduce en su tabla la dupla compuesta por la dirección MAC origen del paquete y puerto de entrada, luego busca la dirección MAC destino del paquete en la tabla y, utiliza el puerto asociado a dicha dirección MAC como puerto de salida. Este proceso se ilustra en la figura 3.32b.

El camino queda confirmado una vez el paquete *ARP-Reply* llega al origen del flujo, esto se observa en la figura 3.32c.



(a) Fase de exploración de la red

(b) Fase de confirmación del camino

(c) Camino final confirmado

Figura 3.32: Fases del protocolo ARP-Path

Capítulo 4

Mecanismo híbrido de recuperación frente a fallos

Es altamente recomendable que todos los protocolos encargados de realizar el reenvío de cada una de las tramas que recorren la red dispongan de, al menos, un mecanismo de protección ante fallos eventuales o, lo que es lo mismo, automatismos que permitan recuperar las comunicaciones ante cualquier tipo de contingencia. Tradicionalmente, los mecanismos encargados de la recuperación se pueden dividir en dos grandes grupos: mecanismos *reactivos* y mecanismos *proactivos*. Los mecanismos de recuperación *reactivos* son aquellos que ejecutan todo su proceso una vez ha ocurrido el fallo, sin necesidad de realizar ningún tipo de acción antes del error. Por contra, los mecanismos *proactivos* son todos aquellos que necesitan realizar algún tipo de acción antes de que ocurra el fallo como, por ejemplo, reservar a priori una serie de recursos de la red o tener calculado un camino alternativo previamente.

Por otro lado, los distintos avances realizados dentro del campo de las redes de comunicación han logrado la creación y definición de un nuevo paradigma arquitectónico, la arquitectura SDN. Este nuevo paradigma rompe con el esquema establecido por la arquitectura tradicional, ya que separa el *Plano de Control* del *Plano de Datos*. La arquitectura SDN, como se ha visto en secciones anteriores, define un *Plano de Control* centralizado que se encarga de conocer, orquestar, controlar y modificar a los elementos que componen la red. El *Plano de Control* es capaz de conocer el estado de la red y de modificar cada uno de elementos que la conforman a cambio de un coste computacional, el cual crece según el tamaño y eventos que deben ser gestionados en la red. Además, en caso de fallos, el número de eventos recibidos para su procesamiento puede aumentar significativamente la carga de trabajo del *Plano de Control*, lo cual, en algunos casos, puede llegar a suponer un problema de escalabilidad importante. Por ello, esta tesis propone una solución innovadora de mecanismo de recuperación caracterizado por ser reactivo, no tener una reserva previa de recursos y utilizar la exploración de la red como método para restablecer la comunicación entre dos nodos usados en redes tradicionales

no SDN. Estas dos últimas características permiten al nuevo sistema de recuperación, por un lado, utilizar el conocimiento que tiene el controlador SDN sobre el estado y la topología de red, y, además, beneficiarse de la versatilidad y potencia de los mecanismos de exploración de red de los sistemas distribuidos. El nuevo sistema de recuperación fue publicado en la contribución [11] y en la patente [16].

4.1 Basic OpenFlow Software Switch (BOFUSS)

En la actualidad, existen multitud de elementos (*hardware* o *software*) compatibles con la arquitectura SDN y, por lo tanto, con sus controladores y protocolos de comunicación. Para la realización de esta tesis, entre todos los dispositivos de red disponibles, se va a hacer uso de switches SDN compatibles con el protocolo OpenFlow. Del mismo modo, esta tesis quiere destacar de entre todos los switches *software* disponibles actualmente *OvS* y *BOFUSS*. La razón por la que esta tesis destaca estos dos switches *software* es porque se trata de los dos dispositivos existentes más utilizados actualmente.

OvS, desarrollado por *VMware*, destaca por su gran rendimiento y su alta integración en el *Kernel* de los sistemas operativos *Linux*. Sin embargo, esto provoca que su codificación sea compleja, por lo que el tiempo necesario para realizar cualquier tipo de modificación suele ser elevado y el desarrollo necesario para realizar dicha modificación muy costoso.

Por su parte BOFUSS [40], desarrollado e implementado en *El Centro de Investigación y Desarrollo de Telecomunicaciones* (*CPqD*) de Brasil, se caracteriza por estar desarrollado siguiendo, de la manera más fiel posible, la especificación establecida dentro del protocolo *OpenFlow 1.3*, por no proporcionar un alto rendimiento y por permitir realizar desarrollos de prototipos de manera rápida y sencilla debido a que la complejidad en su código no es muy elevada.

En conclusión y tras presentar las características más relevantes que tienen OvS y BOFUSS en los párrafos anteriores. Esta tesis ha decidido, dada la diferencia de complejidad existente entre el código de OvS y el de BOFUSS, usar como elemento base para el desarrollo del prototipo del nuevo mecanismos de recuperación híbrido al switch *software* BOFUSS, por su sencillez y el bajo coste necesario para llevar a cabo cualquier modificación en su código.

4.1.1 Estudio del rendimiento de los switches software OvS y BOFUSS

Aunque es bien conocida la diferencia existente entre ambos switches *software*, en esta tesis se ha realizado un pequeño estudio sobre el rendimiento, en términos de BandWidth (BW) y Flow Completion Time (FCT), de cada uno de los switches *software* anteriores. Este estudio se ha publicado en [15].

Switch		Ancho de Banda Máximo (Gbps)	Tiempo de Procesamiento (ms)
OvS	$\overline{x}$	51.413	724.16
	σ	2.6784	32.23
BOFUSS Mejorado	$\overline{x}$	1.184	2354.48
	σ	$3{,}945 * 10^{-3}$	1,4599
BOFUSS Original	$\overline{x}$	0.186	59115.75
	σ	$6{,}86 * 10^{-5}$	130,88

Tabla 4.1: Resultados de Ancho de banda máximo y tiempo de procesamiento en OvS, BOFUSS Original y BOFUSS Mejorado

El estudio de rendimiento se inicia con la realización de una prueba cuyo objetivo es, por un lado, medir el ancho de banda máximo que puede alcanzar cada uno de ellos (OvS y BOFUSS) y, por otro lado, medir el retardo de procesamiento introducido por cada switch *software*. Se quiere destacar que, para la realización de este estudio, se han tomado medidas utilizando OvS y las dos variantes más relevantes de BOFUSS. La primera, denominada como *BOFUSS Original*, es la implementación original realizada en *CPqD*, y la segunda, denominada *BOFUSS Mejorado*, es el desarrollo realizado dentro del marco definido por el proyecto *BEBA*, cuyo objetivo era aumentar el rendimiento de *BOFUSS Original*.

La plataforma *hardware* utilizada para la realización de esta primera prueba está compuesta por un único ordenador con procesador Intel(R) Core (TM) i7 (3,4 GHz) y 24 GBytes de Memoria *RAM*. Este equipo dispone de Ubuntu 14.04 como sistema operativo y Mininet [82] como herramienta de emulación de red.

Además, se utilizó una topología lineal compuesta por dos equipos finales (hosts) interconectados por un único switch *software*. La prueba consistió en el envío de un flujo de gran tamaño utilizando como generador de flujos la herramienta *Iperf*. Esta prueba se repitió 10 veces para conseguir un intervalo de confianza del 95 %.

Los resultados de esta primera prueba se muestran en la tabla 4.1 [15]. En ella se puede observar que, como era de esperar, el rendimiento de OvS es muchísimo mayor que el rendimiento ofrecido por cualquiera de las implementaciones del switch *software* BOFUSS (*BOFUSS Original* y *BOFUSS Mejorado*) . Esta diferencia viene dada, principalmente, por una decisión inicial de diseño que cada equipo de trabajo ha llevado a cabo para implementar cada uno de los switches *software*. La implementación realizada para la creación de BOFUSS sitúa el proceso del propio switch *software* en el *espacio de usuario* del sistema operativo, lo que produce que tenga un mayor número de interrupciones que cualquier aplicación situada en el *espacio de Kernel* y, por lo tanto, aumenta el tiempo que necesita el switch *software* para procesar las tramas, bajando mucho su rendimiento. Por su parte, OvS, divide en dos partes su implementación, una situada en el *espacio de usuario* y la otra en *espacio de Kernel*. La parte situada en el *espacio de usuario* se utiliza principalmente cuando OvS necesita comunicarse con el *Plano de Control*, mientras que, la parte situada en el *espacio de Kernel* es la encargada de procesar cada una de las tramas que llegan al switch *software*, lo que provoca una mejora en su rendimiento debido a que

el procesamiento de las tramas apenas se ve interrumpido y, por lo tanto, el tiempo que tarda una trama en ser encaminada no se vea afectado.

En la misma tabla, también se observa como *BOFUSS Mejorado* mejora en un orden de magnitud el rendimiento de su predecesor. Esto se debe a que una de las modificaciones realizadas consiste en que el switch *software* esté consultando en todo momento si existe alguna trama en cola, lo que provoca un descenso del tiempo de espera de las tramas en la cola y, por lo tanto, un aumento en el rendimiento general del dispositivo.

La segunda medida realizada para estudiar el rendimiento de cada uno de los switches *software* es la demanda de capacidad de proceso que necesitan utilizar cada uno de ellos. Para ello, se ha utilizado la misma plataforma *hardware* anterior, la misma herramienta de emulación y la misma topología, pero no el mismo BW. En esta prueba, a diferencia de la prueba anterior en la que no existía limitación en el BW del flujo, se va han utilizado diferentes BW. El BW tiene un valor inicial de 1 Mbps y, posteriormente, va aumentando tal y como indica el eje de abscisas de la figura 4.1 [15] hasta llegar al valor máximo soportado por cada switch *software* (representado como *MAX*). Cada uno de estos lanzamientos se ha repetido un total de 10 veces para asegurar la validez de la medida y obtener un intervalo de confianza del 95 %. Los resultados obtenidos por esta segunda prueba se muestran en la figura 4.1 [15]. En ella se observa como la demanda de capacidad de proceso, tanto de *OvS* como de *BOFUSS Original*, crece con el BW del flujo. A diferencia de lo que ocurre con *BOFUSS Mejorado*, el cual siempre mantiene constante el uso del procesador (12.5 %). Ese valor se mantiene constante debido a que *BOFUSS Mejorado* está programado para consultar en todo momento si existe o no una nueva trama en alguna de sus colas de entrada, lo que provoca que utilice el 100 % de uno de los ocho núcleos (*Cores*) disponibles en el procesador (12.5 % del total procesador) y, por otro lado, al igual que BOFUSS Original, se ejecuta en un proceso *monohilo*, es decir, solo puede utilizar un único núcleo de todos los disponibles en el procesador.

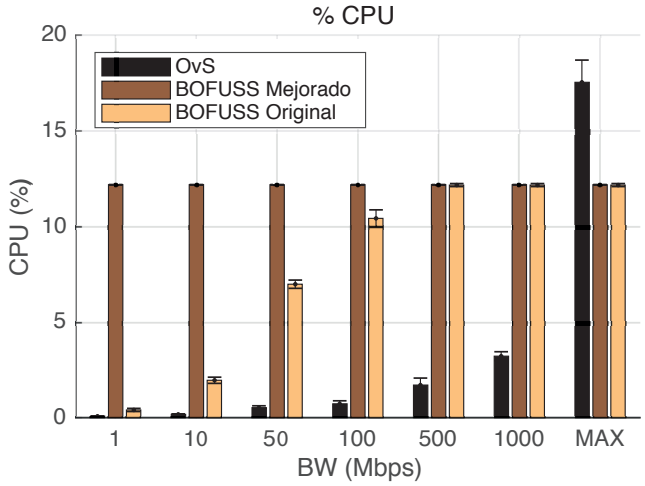


Figura 4.1: Uso del procesador por parte de OvS, BOFUSS Original y BOFUSS Mejorado [15]

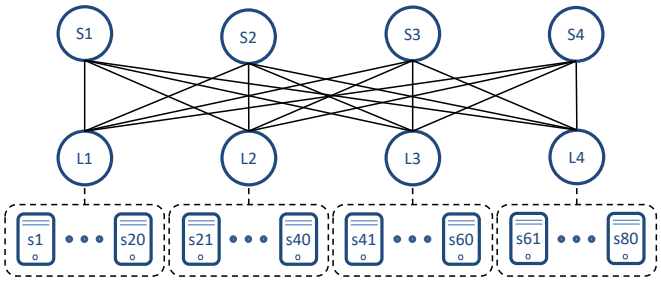


Figura 4.2: Topología Spine-Leaf 4x4x20

Los resultados también confirman la capacidad que tiene OvS de utilizar más de un núcleo del procesador cuando recibe un flujo cuyo BW es muy elevado (*MAX* en la figura 4.1 [15]). Este hecho se observa, claramente, en el resultado obtenido por OvS para el caso BW MAX, ya que en dicho caso, el porcentaje de procesador utilizado por OvS supera el 12.5 % del total del procesador, es decir, OvS está utilizando dos núcleos de forma paralela.

Como última prueba, se ha realizado una comparación en términos de BW y FCT bajo diferentes cargas en la red entre los dos switches *software*. Para ello, se ha utilizado una topología *Spine-Leaf* compuesta por 4 switches *Spine*, 4 switches *Leaf*, 20 equipos por *Leaf* y enlaces cuya velocidad de transmisión se ha fijado a 100 Mbps (ver figura 4.2 [12]) junto a una matriz de tráfico aleatoria, cuya única condición de decisión es que los dos equipos finales elegidos para realizar la comunicación deben estar en diferentes *Leaf*. Esta condición se ha definido así para provocar que la carga en la red se sitúe especialmente en su núcleo y no en los enlaces de acceso. Se han utilizado, también, dos tipos diferentes de distribuciones de tamaño de flujos, *Web Search* [83] y *Data Mining* [84]. Estas distribuciones de tamaño de flujo provienen de datos reales tomados en centros de datos. La figura 4.3 [12] muestra las Funciones de Distribución (Cumulative Distribution Functions (CDFs)) de ambas (*Web Search* y *Data Mining*). En ella, se observa como los flujos han sido clasificados según su tamaño. Los flujos cuyo tamaño sea inferior a 10 Kbyte se han identificado como flujos de tipo *Ratón* (Mouse), los flujos medianos, cuyo tamaño se sitúe entre 10 Kbyte y 10 Mbyte, han sido denominados como flujos de tipo *Conejo* (Rabbit) y, por último, los flujos grandes, cuyo tamaño sea mayor a 10 Mbyte, se han denominado como flujos de tipo *Elefante* (Elephant).

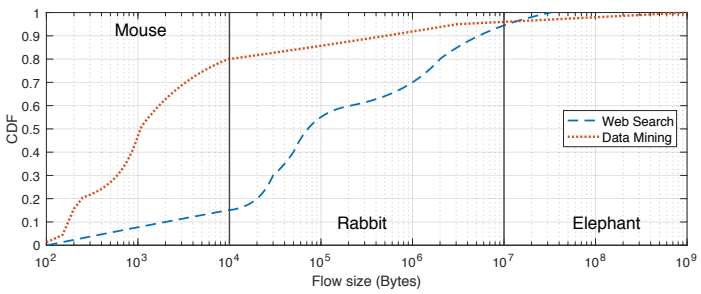


Figura 4.3: Distribución de tamaño de flujos

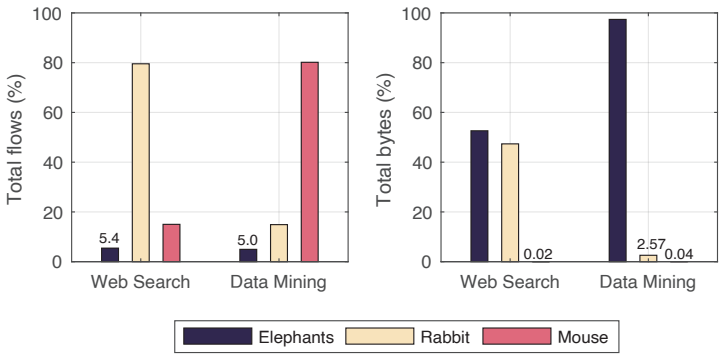


Figura 4.4: Distribución porcentual de número total de flujos vs número total de *bytes* por flujo

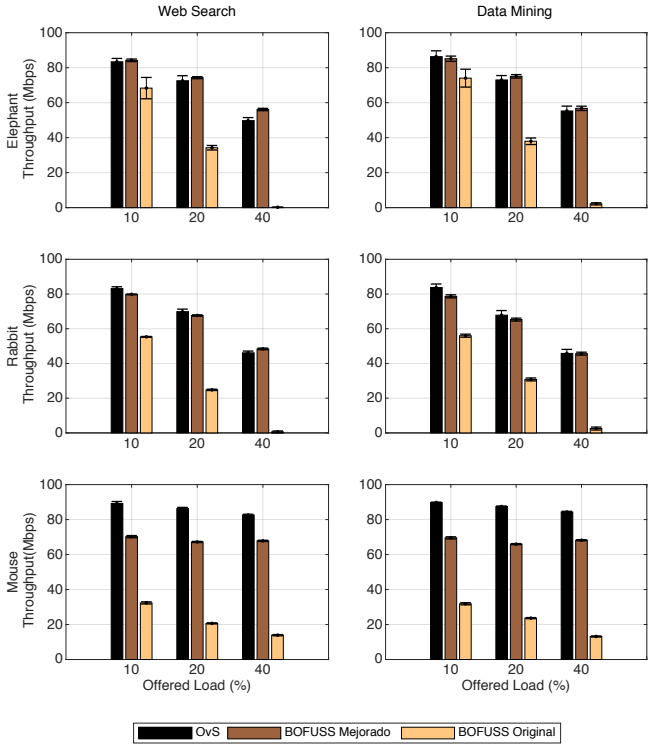
La relación entre el número de flujos de cada tipo en cada una de las distribuciones y de su tamaño en *bytes* queda reflejado en la figura 4.4 [12]. En esta figura podemos observar, por un lado, que para el caso de la distribución *Web Search*, el número de flujos de tipo *Conejo* absorbe el 80 % del número total de flujos, pero aun siendo mucho mayor que el número de flujos de tipo *Elefante* (5.4 % del número total de flujos), su tamaño en *bytes* es inferior a la cantidad de *bytes* que componen los flujos de tipo *Elefante*. Y, por el otro lado, para el caso de la distribución *Data Mining*, el número total de flujos de tipo *Ratón* se sitúa en torno al 80 % del número total de flujos emitidos, mientras que el total de *bytes* utilizados por ellos (0,04 %) es considerablemente inferior al total de *bytes* usados por los flujos de tipo *Elefante*, que se sitúa alrededor del 97 % del total de *bytes* transmitidos. Como resumen de las configuraciones establecidas para la prueba se ha creado la tabla 4.2.

Por último, para la realización de esta prueba ha sido necesario el uso de Mininet como herramienta de emulación, además de una infraestructura formada por 5 ordenadores con procesador Intel(R) Core (TM) i7 y 24 Gbytes de Memoria RAM, todos ellos interconectados mediante un *switch* cuya velocidad de transmisión máxima por puerto es de 1 Gbps.

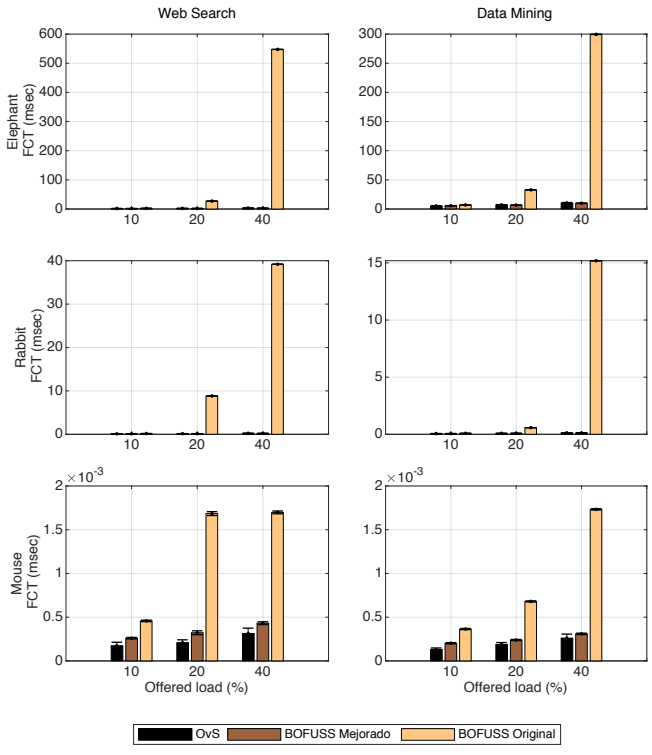
Las pruebas descritas anteriormente dieron como resultados los datos mostrados en las figuras 4.5a [15] y 4.5b [15]. La figura 4.5a [15] representa el BW medio de los

Característica	Valor
Topología de la red	Spine-Leaf (4 - 4) [85]
Equipos por Leaf	20
Matriz de Tráfico	Aleatoria entre *Leaf*
Distribuciones tamaño de flujos	Web Search [83] & Data Mining [84]
Carga ofrecida a la Red (%)	10, 20 & 40 %
Velocidad de los enlaces	100 Mbps
Tiempo de prueba (s)	1800 s
Tiempo transitorio (s)	800 s
Número de repeticiones	10

Tabla 4.2: Configuración para la comparación de rendimiento entre OvS, BOFUSS Original y BOFUSS Mejorado



(a) Comparación en términos del BW medio



(b) Comparación en términos de FCT

Figura 4.5: Comparación del rendimiento entre OvS, BOFUSS Original y BOFUSS Mejorado [15]

diferentes flujos, mientras que la figura 4.5b [15] representa el FCT medio de los mismos. Las gráficas representadas en ambas figuras se ordenan en dos columnas y tres filas: las columnas indican las distribuciones de tamaño de los flujos (*Web Search* en la columna de la izquierda y *Data Mining* en la columna de la derecha), mientras que las filas se ordenan siguiendo el tamaño de los flujos, situando en la fila superior aquellos flujos cuyo tamaño es mayor, en la fila central aquellos flujos cuyo tamaño es mediano y en la fila inferior el tráfico con menor tamaño, es decir, los flujos de tipo *Elefante* se representan en la primera fila, los flujos del tipo *Conejo* en la segunda fila y los flujos del tipo *Ratón* en la tercera y última fila.

En dichas imágenes (figuras 4.5a [15] y 4.5b [15]) se observa como el rendimiento obtenido por OvS, independientemente del tamaño de flujo, es superior al obtenido por cualquiera de dos implementaciones de BOFUSS, así como, un FCT mucho menor que el obtenido por ellas. Esta diferencia de rendimiento se debe a las razones anteriormente expuestas: su mejor codificación, su integración en el *Kernel* de *Linux* y su bajo tiempo de procesamiento. En las mismas figuras se observa como los resultados obtenidos por BOFUSS *Mejorado* sigue manteniendo una gran diferencia de rendimiento (un orden de magnitud) respecto al BOFUSS *Original*. Esta diferencia se explica del mismo modo que se ha hecho en las pruebas anteriores, la nueva versión reduce el tiempo que las tramas permanecen en el switch *software*, ya que en todo momento se está preguntando al sistema si existe algún paquete en las colas de entrada para ser tratado.

4.2 Switch híbrido ARP-Path/OpenFlow Software Switch (AOSS)

Para la puesta en marcha del mecanismo de recuperación híbrido se ha tomado como punto de partida el desarrollo del switch híbrido AOSS realizado en [43]. AOSS, a su vez, está basado en la implementación original del switch *software BOFUSS* [40].

La gran diferencia entre ambos es que *BOFUSS*, al funcionar siguiendo la especificación del protocolo *OpenFlow 1.3* (ver 3.17), tiene definido que todo flujo desconocido para él, o lo que es lo mismo, todo flujo que no disponga de al menos una regla de actuación dentro de las tablas de flujos, sea descartado. Sin embargo, AOSS elimina de su diagrama de flujo esta posibilidad puesto que es capaz de seleccionar caminos de forma autónoma mediante el protocolo ARP-Path [86]. Dicho de otro modo, dispone de una lógica capaz de seleccionar por qué puerto de salida ha de reenviar una trama para hacerla llegar a su destino final de forma autónoma, sin necesidad de consultar al controlador.

Para soportar esta nueva característica se necesitó realizar un cambio dentro del diagrama de flujo del switch original. Dicho cambio consistió en eliminar la función encargada de descartar las tramas en el caso de no existir una regla e insertar la lógica del protocolo ARP-Path. Este cambio se puede observar en la figura 4.6 [43]. Este desarrollo,

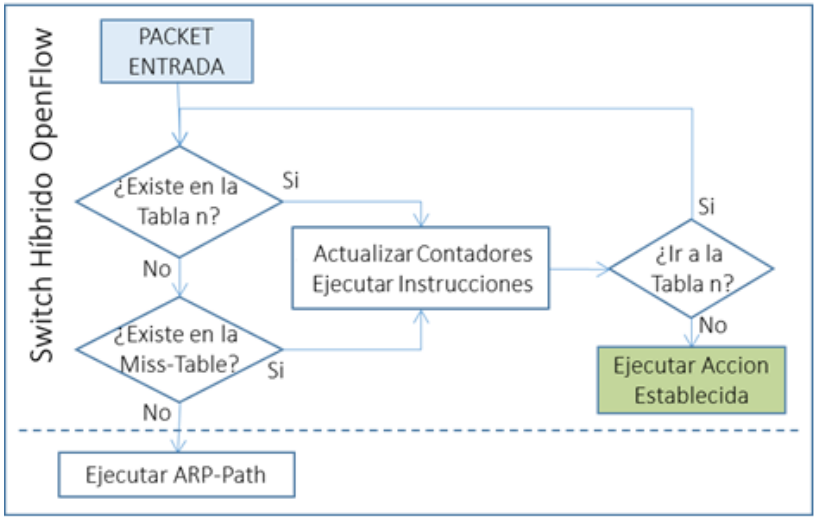


Figura 4.6: Diagrama de flujo switch AOSS

además de dotar al switch de la capacidad de seleccionar caminos de forma autónoma, logró una reducción en el tiempo de establecimiento de los caminos y una reducción del procesamiento del *Plano de Control*, lo que produjo una mejora en el rendimiento de éste. El aumento de rendimiento viene dado porque se dispone de más recursos para realizar otras tareas de mayor prioridad, tales como, manejar los flujos considerados como *premium*, mejorar la seguridad de la red o, simplemente, monitorizar la red en búsqueda de anomalías.

AOSS, a diferencia de otras versiones de switches híbridos que se pueden encontrar en el mercado actual, es capaz de utilizar todos sus puertos independientemente del paradigma de red conectado a dicho puerto. Actualmente la mayor parte de los switches híbridos comerciales necesitan tener configurado el modo de trabajo (tradicional o SDN) de cada uno de los puertos. Esta configuración provoca que en la mayoría de los casos el dispositivo se utilice como punto de interconexión entre dos redes con arquitecturas diferentes.

Por otro lado, y aprovechando que se va a presentar un nuevo mecanismo de recuperación, cabe citar los dos mecanismos de recuperación existentes previamente. El primero de ellos funciona de manera distribuida, es decir, sin necesidad de tener que solicitar al controlador un nuevo camino, ya que utilizando dos procesos de exploración de red junto al sistema de bloqueo definido por el protocolo ARP-Path logra restaurar la comunicación. El segundo mecanismo se realiza de manera centralizada, es decir, aprovecha la capacidad de AOSS para comunicarse con el controlador para restablecer la comunicación. En este caso, AOSS solicita de forma explícita una nueva ruta al controlador para los flujos afectados por el fallo. En resumen, AOSS dispone de dos recuperaciones distintas ya implementadas en su *software*, una apoyada en controlador (*Recuperación Centralizada*) y otra apoyada exclusivamente en la red (*Recuperación Distribuida*).

4.3 Recuperación híbrida

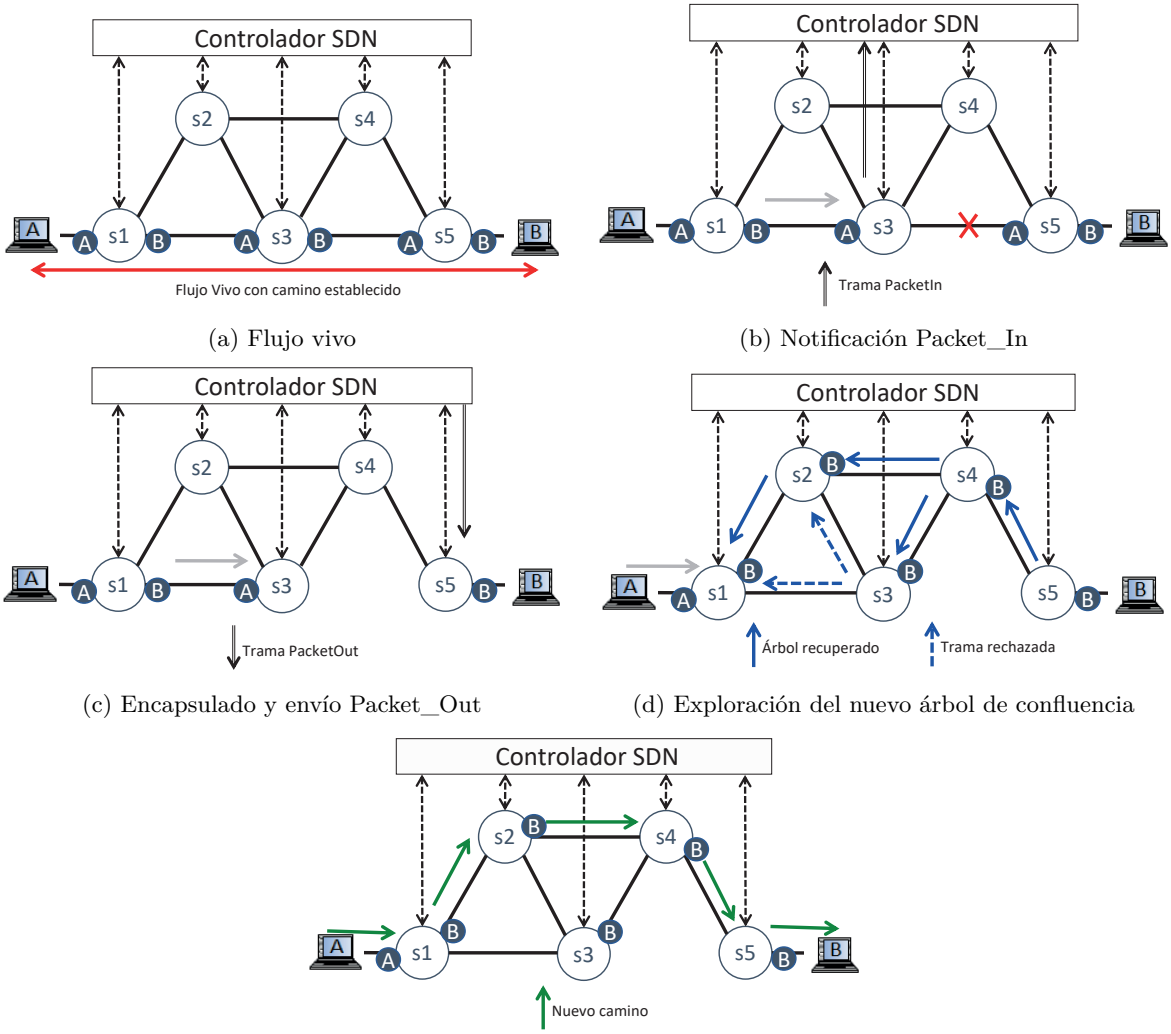
En esta sección se presenta un mecanismo de recuperación reactivo capaz de, ante un fallo eventual en la red (caída de un enlace o un nodo), recuperar los caminos dañados de los distintos flujos que atraviesan los switches o enlaces caídos, aprovechado las características híbridas del switch. Para ello, se hace uso de forma colaborativa, por un lado, de la capacidad que tiene el controlador de conocer y modificar la red y, por otro lado, de la potencia y versatilidad que ofrece el mecanismo de exploración y descubrimiento de caminos de AOSS.

Este mecanismo se implementa en dos elementos de red diferentes: en los switches AOSS de la capa de infraestructura y en el controlador perteneciente a la capa de control. El primero, el switch *software* híbrido SDN (AOSS), ya que al tratarse de un mecanismo de recuperación híbrido se necesita que los switches de la red sean capaces de gestionar, procesar y actuar siguiendo los distintos métodos establecidos por dicho mecanismo. Para ello, el switch debe ser capaz de manejar un tipo especial de trama definida por el mecanismo, y cuya función es la exploración de un nuevo árbol de confluencia cuya raíz es el switch destino final del flujo original y sus ramas definirán los nuevos caminos para llegar a él. El segundo elemento de la red al que se le debe introducir la lógica del proceso de recuperación es al *Plano de Control*, que por simplicidad y para el estudio llevado a cabo en esta tesis estará compuesto únicamente por una única instancia o controlador. Para ello, se ha desarrollado una aplicación encargada de seleccionar el punto de la red adecuado para el inicio de la recuperación, discernir entre peticiones de recuperación repetidas y procesar el paquete especial de recuperación.

4.3.1 Mecanismo de recuperación

Para explicar el funcionamiento general del mecanismo de recuperación se va a hacer uso del ejemplo mostrado en la figura 4.7 [16]). Compuesto por una topología formada por cinco switches híbridos interconectados como indica la figura 4.7a. Suponemos, también, que existe un flujo vivo con origen el equipo A y destino el equipo B, y que dicho flujo sigue la ruta marcada en rojo en la figura. A partir de un instante determinado el enlace que une a los nodos *s3* y *s5* se ve afectado por una caída y, por lo tanto, el camino entre el equipo A y el equipo B se rompe.

Posteriormente, en el momento en que el nodo *s3* detecta la caída del enlace, se lo comunica al controlador a través de un paquete *Port_Status* perteneciente al protocolo *OpenFlow*. Este paquete indica al controlador que al menos un puerto de dicho switch ha sido modificado. Este hecho, además, es usado por el mecanismo de recuperación propuesto en esta tesis como mecanismo de prevención de posibles ataques de denegación de servicio ante la llegada de falsas solicitudes de recuperación. A continuación, el switch espera a recibir una trama, cuyo puerto de salida sea el puerto afectado por el fallo,



(a) Flujo vivo

(b) Notificación Packet_In

(c) Encapsulado y envío Packet_Out

(d) Exploración del nuevo árbol de confluencia

(e) Flujo recuperado correctamente

Figura 4.7: Recuperación híbrida

para una vez recibido dicha trama encapsularla en un paquete *Packet_In* y enviársela al controlador (ver figura 4.7b). Este mensaje lleva la información suficiente para que el controlador pueda comenzar con el mecanismo de recuperación. Este proceso lo repite de forma idéntica el nodo *s5*, puesto que el enlace roto afecta a los dos nodos y, por lo tanto, los dos deben solicitar una recuperación.

A continuación, el controlador analiza el paquete *Packet_In* recibido para obtener de él los siguientes datos: el Datapath Id (DpId) del switch que solicita la recuperación, la dirección MAC destino del flujo, la dirección MAC origen del flujo y, por último, el *EtherType* de la trama. Una vez conocida la dirección MAC destino, el controlador realiza una búsqueda dentro de la tabla que relaciona cada uno de los equipos finales con los switches pertenecientes a la red, a partir de ahora *tabla de vecinos* (esta tabla es explica en más en profundidad en la sección 4.3.4.2), para obtener el *DpId* del switch al que dicho equipo está directamente conectado. Se entiende como 'directamente conectado' una de

las dos siguientes situaciones: una conexión directa entre el switch y el equipo final, o cuando el switch actúe como punto de conexión entre dos redes estando el equipo final situado en la red externa. En cualquiera caso, siempre se ha de cumplir que el switch debe ser el primer nodo de la red en recibir un paquete ARP procedente del equipo. Una vez el controlador ha localizado dicho nodo, crea una trama *Path Request*. En caso contrario, al no encontrar un switch conectado directamente con el equipo final, descarta la petición de recuperación. La trama de recuperación, *Path Request*, se encapsula en un paquete *Packet_Out* perteneciente al conjunto de paquetes definidos en el protocolo *OpenFlow* para, posteriormente, ser enviada al switch destino seleccionado en el paso anterior. Una vez el paquete es recibido por el switch, éste lo desencapsula para obtener la trama *Path Request* original. Posteriormente, el switch difunde la trama original por todos sus puertos para comenzar de este modo el proceso de exploración del nuevo árbol de confluencia. Este proceso se puede observar en la figura 4.7c.

El proceso de recuperación termina cuando el flujo original, una vez difundida la trama *Path Request* y creado el árbol de confluencia con raíz el switch frontera destino del flujo original, confirma una de las ramas. Para ello, las tramas del flujo original avanzan por el árbol de confluencia hacia su raíz (ver figura 4.7d). Este mecanismo de recuperación, para evitar bucles a la hora de generar el árbol de confluencia, hace uso del mecanismo de bloqueo definido en el protocolo *ARP-Path* [44]. Este mecanismo establece que un switch cualquiera debe descartar toda copia repetida de una trama de difusión que llegue al switch por un puerto distinto al puerto de entrada de la primera copia recibida de la trama. Este proceso de descarte se realizará únicamente durante un corto espacio de tiempo (denominado tiempo de bloqueo del puerto) desde la primera recepción del paquete.

4.3.2 Trama Path Recovery

El mecanismo presentado en esta sección, como se ha visto en el apartado 4.3.1, necesita de una nueva trama específica para la recuperación. Dicha trama ya ha sido presentada en la explicación del proceso de recuperación como *Path Recovery*, se la ha denominado así debido a su objetivo. Esta trama está formada únicamente por una cabecera del tipo *Ethernet* y un *payload* cuyo objetivo es completar el tamaño mínimo exigido por las redes *Ethernet* actuales.

La cabecera *Ethernet* está formada, a su vez, por una dirección MAC destino de difusión (en el caso de las pruebas realizadas una dirección Broadcast(*FF:FF:FF:FF:FF:FF*), pero también admite direcciones multicast), una dirección MAC origen, la cual toma el valor de la dirección MAC destino del flujo que ha sufrido la rotura en su camino y, por último, un *Ethertype* especial asignado específicamente para este mecanismo. La estructura completa del paquete se ha representado en la figura 4.8.

MAC Destino	MAC Origen	EtherType	PayLoad
FF:FF:FF:FF:FF	Dirección MAC Equipo Destino	EtherType asignado a recuperación	Payload de relleno

Figura 4.8: Estructura y Contenido de la trama *Path Recovery*

4.3.3 Diseño de la función realizada por el controlador SDN

En concordancia con lo expuesto en el apartado 4.3.1, este mecanismo de recuperación necesita apoyarse en un controlador, ya que aprovecha su conocimiento de la topología para implementar parte del mecanismo de recuperación.

En la figura 4.9 se muestra el diagrama de flujo que define el comportamiento de la aplicación del controlador para realizar el proceso de recuperación. El proceso comienza cuando un *Packet_In* llega al controlador y se extrae: el DpId del switch que envío el paquete y la dirección MAC destino de la trama original encapsulada dentro del *Packet_In*. A continuación, el controlador confirma que la trama recibida es una solicitud de recuperación válida, comprobando primero si ha recibido un mensaje *Port_Status* procedente del mismo switch. Para ello, verifica que el DpId del switch que envió los paquetes es el mismo y, después, constata si el puerto de salida de la trama recibida mediante el *Packet_In* ha sido modificado. Para esto último, debe cerciorarse que el puerto modificado en el switch (notificado al controlador a través del mensaje *Port_Status*) coincide con el puerto de salida asignado para la trama recibida. Si ambas comprobaciones dan un resultado positivo, la solicitud de recuperación es considerada como válida por parte del controlador. En caso de ser una solicitud válida, el controlador chequea si el equipo destino del flujo está dentro de su lista de equipos finales y, solo en caso afirmativo, busca el DpId del switch conectado directamente con dicho equipo. El controlador genera

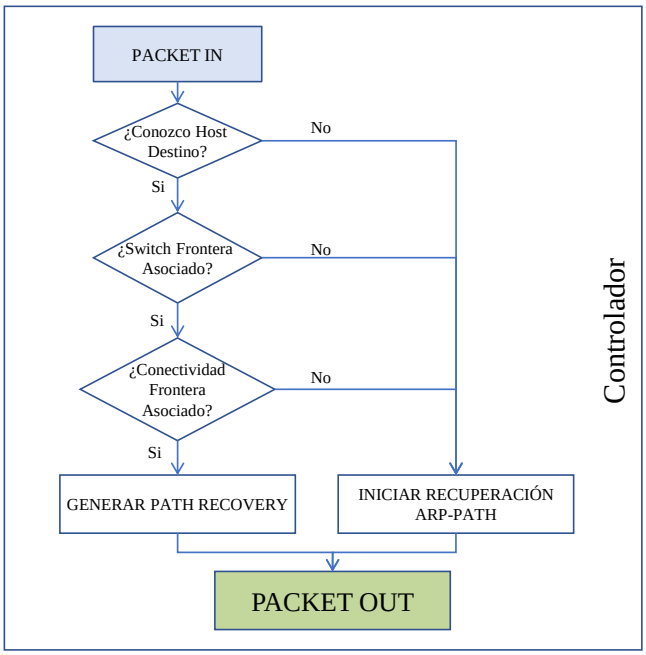


Figura 4.9: Diagrama de flujo del controlador para la recuperación híbrida

una trama *Path Recovery* siguiendo la estructura expuesta en el apartado 4.3.2. En caso contrario, si alguna de las tres condiciones da un resultado negativo, el controlador descarta la solicitud de recuperación.

El mecanismo propuesto en esta tesis también dispone de una función que evita generar múltiples recuperaciones de manera paralela del mismo flujo. Para ello, el controlador, una vez generado la trama *Path Recovery* crea una nueva entrada, en la tabla de recuperaciones activas, indicando en ella la dirección MAC origen del flujo, la dirección MAC destino, *Ethertype* y, en el caso de tratarse de una trama TCP/UDP, puerto origen, puerto destino, todo ello junto a una marca de tiempo (*timestamp*). Esta tabla contendrá la información necesaria para determinar, en el caso de que llegue otra petición de recuperación al controlador, si dicha recuperación está o no activa. Si al realizar una búsqueda, el controlador, se encuentra con un registro cuya marca de tiempo no haya expirado, sabrá que existe una recuperación activa y descartará la petición. En caso contrario, si no existe un resultado válido (no existe registro o el registro tiene una marca de tiempo expirada), el controlador podrá iniciar el mecanismo de recuperación. Esta función se ha creado para evitar ataques de negación de servicio, así como, para reducir el número de recuperaciones activas en la red para evitar su sobrecarga innecesaria.

4.3.4 Implementación en el controlador RYU

La implementación de la funcionalidad en el controlador se ha realizado mediante una división en dos funciones: la primera define la estructura de la nueva trama, su creación y gestión; mientras que la segunda, define las acciones para llevar a cabo el proceso de selección de nodos, comprobación de recuperación activa y envío de la nueva trama.

4.3.4.1 Definición y creación de la trama Path Recovery

Para realizar la gestión de la nueva trama se ha definido una nueva clase donde se recoge la estructura definida en la figura 4.8 para ella, así como el serializador o *deparser* y el deserializador o *parser* de la misma.

El fragmento de código ilustrado en la parte izquierda del código 4.1 representa la estructura de la trama *Path Recovery* codificada en lenguaje *Python* para el controlador *RYU*. En él se observa cómo se define cada uno de los tres campos definidos por el mecanismo para la generación del paquete: la *dirección de difusión*, la dirección MAC destino del flujo y el *EtherType* asignado anteriormente. Del mismo modo, las funciones de serialización (*deparser*) y deserialización (*parser*) se representan mediante las funciones situadas en la parte derecha del código 4.1. Estas funciones son las encargadas de generar la trama de difusión (función *serialize*) y, en el caso de recibir un paquete *Packet_In* con dicha trama encapsulada dentro de él, obtener los parámetros de la misma (función *parser*).

Código 4.1: Codificación de la librería Path Recovery en el controlador

```
class ethernet(packet_base.PacketBase):
    """Ethernet header encoder/decoder class.

    An instance has the following attributes at least.
    MAC addresses are represented as a string like '08:60:6e:7f:74:e7'.
    __init__ takes the corresponding args in this order.

    ============== ================== ====================
    Attribute      Description        Example
    ============== ================== ====================
    Bcast_address  Broadcast Address  'ff:ff:ff:ff:ff:ff'
    src_flow       source address     '08:60:6e:7f:74:e7'
    ethertype      ether type         0x7798
    ============== ================== ====================
    """

    _PACK_STR = '!6s6sH'
    _MIN_LEN = struct.calcsize(_PACK_STR)
    _MIN_PAYLOAD_LEN = 46
    _TYPE = {
        'ascii': [
            'src_flow', 'Bcast_address'
        ]
    }

    def __init__(self, src_flow='00:00:00:00:00:00' ):
        super(ethernet, self).__init__()
        self.Bcast_address = 'ff:ff:ff:ff:ff:ff'
        self.src_flow = src_flow;
        self.ethertype = 0x7798;
```

```
    @classmethod
    def parser(cls, buf):
        Bcast_address, src_flow, ethertype = struct.unpack_from(cls._PACK_STR, buf)
        return (cls(addrconv.mac.bin_to_text(Bcast_address),
                                              addrconv.mac.bin_to_text(src_flow), ethertype
                                          ),
                                  ethernet.get_packet_type(ethertype), buf[ethernet._MIN_LEN:]
                )

    def serialize(self, payload, prev):
        # Append padding if the payload is less than 46 bytes long
        pad_len = self._MIN_PAYLOAD_LEN - len(payload)
        if pad_len > 0:
            payload.extend(b'\x00' * pad_len)

        return struct.pack(ethernet._PACK_STR,
                           addrconv.mac.text_to_bin(self.Bcast_address),
                           addrconv.mac.text_to_bin(self.src_flow),
                           self.ethertype
                           )
```

4.3.4.2 Implementación lógica de la aplicación basada en controlador RYU

La implementación de la aplicación que va a encargarse de gestionar el proceso de recuperación se ha dividido en tres funciones separadas. La primera, representada en el algoritmo 4.1, se encarga de almacenar en la *tabla de vecinos* la relación de vecindad entre switches y equipos finales. La segunda, representada en el algoritmo 4.2, tiene como objetivo la búsqueda del switch conectado directamente con el equipo final. Por último, la función representada por el algoritmo 4.3, que se ocupa de crear y enviar la trama de recuperación al switch destino seleccionado.

Los controladores SDN, para conocer el punto donde cada uno de los equipos finales se conectan a la red, aprovechan la necesidad que tienen los equipos de conocer la dirección física del equipo destino de la comunicación. Este proceso se realiza mediante el intercambio de paquetes propios del protocolo ARP. Este intercambio consiste en que un equipo cualquiera de la red, para obtener la dirección física de otro equipo de la misma red, envía una trama de difusión del tipo *ARP-Request* solicitando la dirección física de dicho equipo, y éste una vez recibe dicha trama contesta con su dirección física mediante el envío de una trama *unicast* del tipo *ARP-Reply*. Cada vez que un switch recibe una de estas tramas manda una copia al controlador para informarle de la existencia de un nuevo equipo final en la red, y es justo en este momento cuando el controlador asocia a los equipos finales con los switches SDN a los que están directamente conectados. Esta asociación se

Data: Paquete Packet_In
Result: Actualizar Tabla Vecinos
begin
  Dirección MAC Equipo Final = Dirección MAC Origen paquete ARP;
  Puerto = In Port Packet_In;
  DpId Nodo = DpId Packet_In;
  if *NOT Dirección MAC destino in tabla de vecinos* **then**
    Insertar Tupla (MAC Equipo Final | DpId Nodo | Puerto | Marca Temporal);
  Gestionar Packet_In;

Algoritmo 4.1: Gestión de la *tabla de vecinos*

```
Data: Paquete Packet_In
Result: DpId del nodo frontera del flujo
begin
    if Dirección MAC destino in tabla de vecinos then
        return DpId Nodo;
    else
        return NULL;
```

Algoritmo 4.2: Búsqueda del nodo frontera del flujo

```
Data: Paquete Packet In
Result: Inicialización de la recuperación
begin
    DpId Nodo = Búsqueda MAC;
    if DpId Nodo != NULL then
        MAC Origen Recuperación = MAC Destino Flujo;
        Nodo Destino flujo = DpId Nodo;
        Creación Paquete Recuperación (MAC Origen Recuperación);
        Envío Paquete Recuperación (DpId Nodo, Flood);
    else
        Descartar Recuperación
```

Algoritmo 4.3: Creación y envío de una trama de recuperación

realiza de una manera muy sencilla, el equipo final está directamente conectado con el switch que envió al controlador la primera trama ARP cuya dirección MAC origen coincida con la dirección física del equipo final. Esta lógica se puede observar en el algoritmo 4.1. En dicho algoritmo se muestra como el controlador comprueba previamente si existe una entrada con la dirección física del equipo final destino dentro de la *tabla de vecinos*. En caso de no existir, el controlador crea una nueva entrada en su tabla vinculando dicha dirección física con el DpId del switch que envió el *Packet_In*. Posteriormente, reenvía la trama ARP al switch encapsulándola en un *Packet_Out* e indicando el puerto o puertos por los que el switch debe reenviar la trama.

El controlador, una vez ha recibido una trama de solicitud de recuperación válida, obtiene el *DpId* del switch destino del flujo (algoritmo 4.2) consultando, para ello, su *tabla de vecinos*. Si el *DpId* devuelto no es nulo, es decir, existe una entrada en la *tabla de vecinos* donde se relaciona el equipo final con un switch, el controlador procede a crear la trama *Path Recovery*. Para ello, obtiene de la trama recibida la dirección física del equipo final y la utiliza como dirección MAC origen de la trama *Path Recovery*. A continuación, inserta una dirección de difusión (para el caso de esta tesis la dirección de broadcast *FF:FF:FF:FF:FF:FF*) como dirección MAC destino y, por último, el valor del *EtherType* de la trama lo iguala al código asignado al mecanismo de recuperación. Finalmente encapsula esta nueva trama en un paquete *Packet_Out* y lo envía al switch cuyo DpId coincide con el DpId obtenido en la *tabla de vecinos*. Por el contrario, si el valor del DpId recibido tras la búsqueda en la *tabla de vecinos* fuera un valor nulo, el controlador descarta la recuperación, ya que no dispone de un switch desde el cual iniciar el proceso de recuperación (ver el algoritmo 4.3).

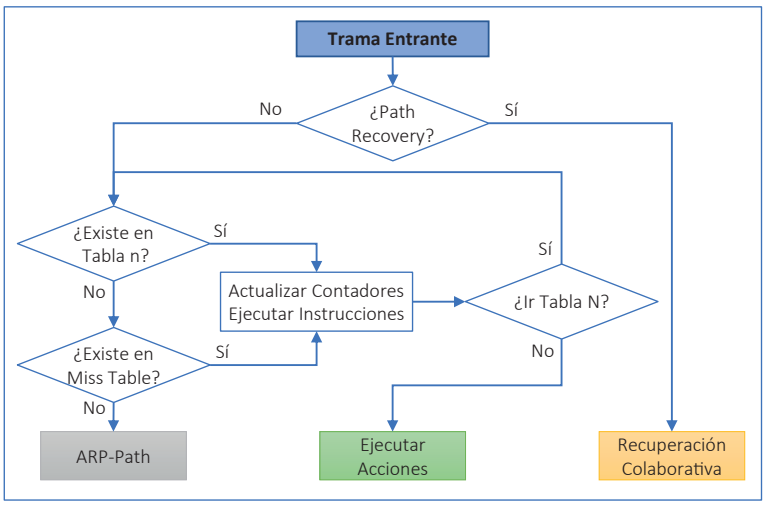


Figura 4.10: Nuevo diagrama de flujo del switch híbrido para introducir el nuevo mecanismo de recuperación

4.3.5 Diseño del mecanismo de recuperación en el conmutador

Como se ha mencionado durante todo el capítulo, el mecanismo de recuperación que se propone hace uso de las características más relevantes de cada uno de los dos paradigmas de red más relevantes hasta el momento (SDN y tradicional). Por lo que, además de realizar cambios en la capa de control de la red SDN, es necesario realizar algunos cambios en el diagrama funcional de AOSS.

El diagrama funcional resultante tras las modificaciones realizadas sobre AOSS se muestra en la figura 4.10. En él, se observa cómo se establece un paso previo a toda la lógica ya existente en AOSS. Este paso es el encargado de detectar la entrada de tramas de recuperación y enviarlas al manejador encargado de la recuperación. Se ha colocado en primer lugar, con el objetivo de dar prioridad a estas nuevas tramas para que cualquier recuperación activa en la red sea lo más rápida posible. Es por ello por lo que el switch, lo primero que va a comprobar una vez reciba cualquier trama, es si dicha trama es del tipo *Path Recovery*. En el caso de ser una trama de recuperación el switch procede a realizar los diferentes procesos establecidos dentro del mecanismo de recuperación (representado en color amarillo en la figura 4.10); en caso contrario, el nodo proseguirá con la lógica original definida en AOSS.

El proceso de recuperación (mostrado en la figura 4.11) se inicia con la adquisición del valor de la dirección física del dispositivo origen incluido en la trama *Path Recovery* (dirección MAC origen de la trama) por parte del switch. Este hecho se justifica porque, por un lado, la trama *Path Recovery* es una trama de difusión, por lo tanto, la dirección MAC destino será una dirección *multicast o broadcast* y, por otro lado, el objetivo de esta trama es explorar la red para crear un nuevo árbol de confluencia con raíz el equipo destino del flujo original que se identificó mediante la dirección física destino de la trama original, la cual fue usada por el controlador para crear el *Path Recovery*.

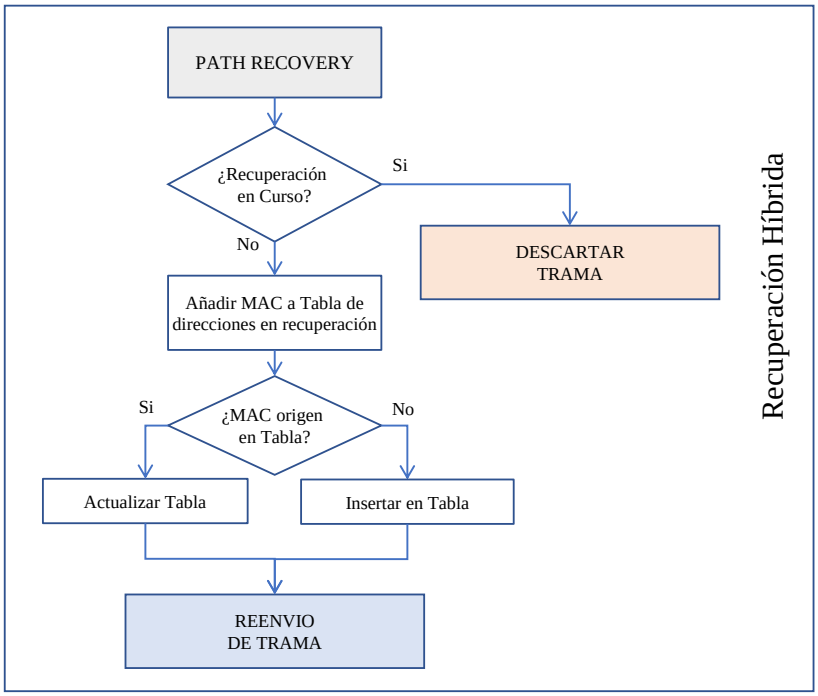


Figura 4.11: Diagrama de flujo de la función de recuperación híbrida

Una vez el controlador dispone de la dirección física del equipo final, pasa a comprobar si existe una recuperación activa, para ello busca la dirección MAC del equipo final dentro de la tabla de recuperaciones. Si la búsqueda devuelve un resultado positivo, es decir, existe un registro cuya marca temporal esté en vigor, el switch pasa a comparar el puerto de entrada, de la trama *Path Recovery* recibida con el valor del puerto de entrada guardado en el registro. En el caso de que los puertos sean diferentes, el switch procede a descartar la trama para evitar bucles durante el proceso de generación del nuevo camino. En caso contrario, es decir, no existe un registro activo en la tabla o el puerto de entrada de la trama *Path Recovery* recibido y el almacenado en el registro sean idénticos, el switch actualiza la tabla mediante la creación de un nuevo registro o la actualización del registro existente, y finalmente, reenvía la trama *Path Recovery* por todos sus puertos menos su puerto de entrada.

4.4 Evaluación

El mecanismo de recuperación propuesto tiene como objetivo principal de restablecer una comunicación entre dos equipos finales. Por este motivo, las pruebas para su evaluación se han diseñado para cumplir con los siguientes objetivos: en primer lugar, comprobar el correcto funcionamiento del mecanismo observando que el proceso realmente restablece la comunicación entre los equipos finales, para ello, en esta prueba se ha medido la cantidad de paquetes perdidos durante el proceso de recuperación de un flujo UDP previamente

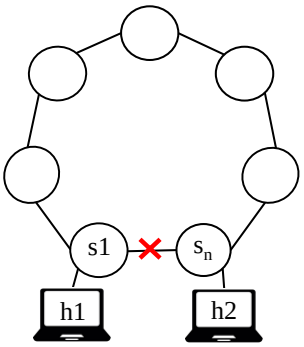


Figura 4.12: Topología en anillo

establecido entre los dos equipos. En segundo lugar, medir el tiempo utilizado por el proceso de recuperación para restablecer dicha comunicación. Y, por último, cuantificar el procesamiento necesario por parte del controlador para llevar a cabo el mecanismo propuesto.

Para realizar esta evaluación se ha utilizado una red con una topología en anillo compuesta por dos equipos finales y siete switches capaces de llevar a cabo el mecanismo de recuperación propuesto. Respecto a los enlaces, se ha decidido establecer una velocidad de transmisión máxima igual a 100 Mbps. La topología completa se muestra en la figura 4.12 [11].

Las pruebas realizadas emulan una caída del enlace que une el switch $S1$ con el switch S_n, que en el momento de su caída, está siendo utilizado para transmitir un flujo UDP con origen el equipo $h1$ y destino el equipo $h2$. El flujo establecido entre dichos equipos se emite a velocidades de transmisión que van desde los 10 Mbps hasta los 80 Mbps, en incrementos de 10 Mbps. Para obtener un mayor realismo en la prueba, se ha fijado en el mismo sentido de la recuperación (desde el nodo S_n al nodo $S1$) un tráfico de fondo con un régimen binario fijado en 50 Mbps, la mitad de la velocidad máxima del enlace.

La razón por la cual se ha utilizado este entorno para llevar a cabo la evaluación es que al situar los equipos finales en switches adyacentes y, una vez establecido el flujo de datos entre ellos, romper el enlace que une dichos switches (enlace $S1$-S_n), fuerza a que él único camino de recuperación posible se establezca atravesando todos y cada uno de los switches que conforman la red y, por lo tanto, es necesario realizar el proceso de recuperación para el caso más adverso que es atravesar todos los dispositivos de la red.

Con respecto a los equipos donde se van a realizar las pruebas, se ha contado con una infraestructura formada por un ordenador con procesador Intel(R) Core (TM) i7 y 24 GBytes de Memoria RAM. Asimismo, se ha utilizado Mininet [82] como herramienta de emulación de red. Al ser una plataforma de emulación se usa la implementación real de la pila TCP, con *Cubic* como control de congestión. El desarrollo de la lógica de controlador se realiza en una aplicación basada en el controlador *RYU* e implementada en *Python*.

Flujo UDP (Mbps)	10	20	30	40	50	60	70	80
Media paquetes perdidos	3.2	7.2	7.2	9.7	13.4	13.7	14.7	16.8
Intervalo de confianza al 95 %	0.3427	0.4197	0.7469	0.9425	0.9541	1.0174	2.5720	0.6412

Tabla 4.3: Media de paquetes perdidos durante el proceso de recuperación

4.5 Resultados

En esta sección se van a presentar y analizar los resultados obtenidos tras las pruebas realizadas. Para un mejor análisis, esta sección se va a dividir en tres subsecciones diferentes. En la primera subsección se procede a analizar los resultados para validar el funcionamiento del mecanismo, en la segunda subsección se analizan los resultados obtenidos tras las medidas temporales y, por último, en la tercera subsección se analizan los resultados adquiridos en el controlador.

4.5.1 Validación del funcionamiento

Tras la prueba para la validación del correcto funcionamiento del mecanismo se han obtenido los resultados mostrados en la tabla 4.3. En dicha tabla se observa como el número de paquetes perdidos durante el proceso de recuperación es pequeño, lo que indica, por un lado, que en un determinado momento la comunicación ha sufrido una caída y, por otro lado, que tras ello el mecanismo de recuperación ha logrado restablecer la comunicación. También se observa como a medida que aumenta la velocidad de transmisión del flujo UDP aumenta el número de paquetes perdidos. Este hecho es algo esperable, ya que, al aumentar la velocidad de transmisión, debe aumentar el número de paquetes enviados por unidad de tiempo, lo que hace que dentro de un mismo intervalo de tiempo el número de paquetes sea mayor y, por lo tanto, se pierdan más paquetes. También, se observa como el intervalo de confianza se mantiene pequeño en todo momento, lo que indica que el procedimiento además de funcionar correctamente lo hace de una manera muy estable.

Por lo tanto, a la vista de los resultados mostrados se puede afirmar que el mecanismo de recuperación funciona correctamente y, por lo tanto, la prueba de validación del funcionamiento ha sido un éxito.

4.5.2 Tiempo de recuperación de la red

Los resultados obtenidos tras las mediciones temporales del proceso de recuperación se muestran en la figura 4.13 [11]. En ella se observa una comparación entre los tres tipos de recuperaciones diferentes existentes en el código de AOSS. La primera, la *Recuperación Distribuida* donde el controlador no tiene ningún tipo de papel y todo el proceso se realiza mediante mecanismos de exploración por difusión de paquetes. Esta recuperación se ha representado mediante la gráfica de color verde y con línea punteada. La segunda, la

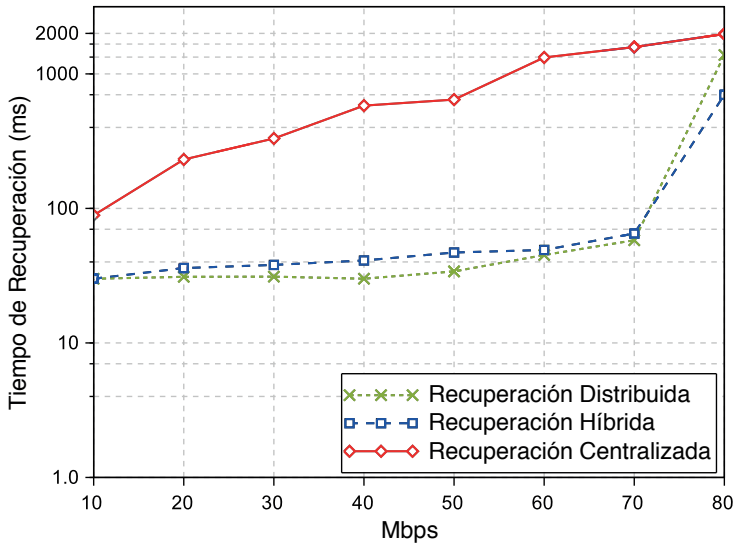


Figura 4.13: Comparación entre los tiempos de recuperación obtenidos con distintos mecanismos [11]

Recuperación Híbrida propuesta en esta tesis representada mediante la gráfica de color azul oscuro y línea discontinua. Y, por último, la *Recuperación Centralizada*, basada únicamente en la computación realizada por parte del controlador y donde los switches simplemente ejecutan las reglas instaladas por él. Esta recuperación se ha representado mediante la gráfica de color rojo y línea continua.

Por un lado, en dicha figura se observa como para el caso de que la red tenga una carga media-baja (desde el 10 % hasta el 70 %), la *Recuperación Centralizada* necesita de un tiempo mayor para recuperar la comunicación que el resto de los mecanismos de recuperación (*Recuperación Híbrida* o *Recuperación Distribuida*). Principalmente, esto se debe a que el proceso de recuperación centralizada necesita un tiempo mayor para modificar cada una de las reglas de reenvío en cada uno de los switches que los otros dos procesos de recuperación. Y, por otro lado, cuando la red está altamente cargada (80 % de carga) el tiempo para realizar los diferentes procesos de recuperación es muy parecido en todos los casos. Del mismo modo, se ve como la *Recuperación Distribuida* y la *Recuperación Híbrida* tienen un tiempo de recuperación muy similar cuando la carga en la red es inferior al 70 %, mientras que en cargas altas (>80 %) la *Recuperación Híbrida* necesita de un tiempo de recuperación algo inferior. Este hecho sucede debido a que, a mayor carga en la red, las colas en los dispositivos son mayores y, por lo tanto, el tiempo necesario para realizar la exploración también es mayor, lo que perjudica más a la *Recuperación Distribuida* al requerir dos exploraciones. Estos tiempos podrían reducirse dando mayor prioridad a este tipo de tramas si se implementará algún mecanismo de QoS a nivel 2.

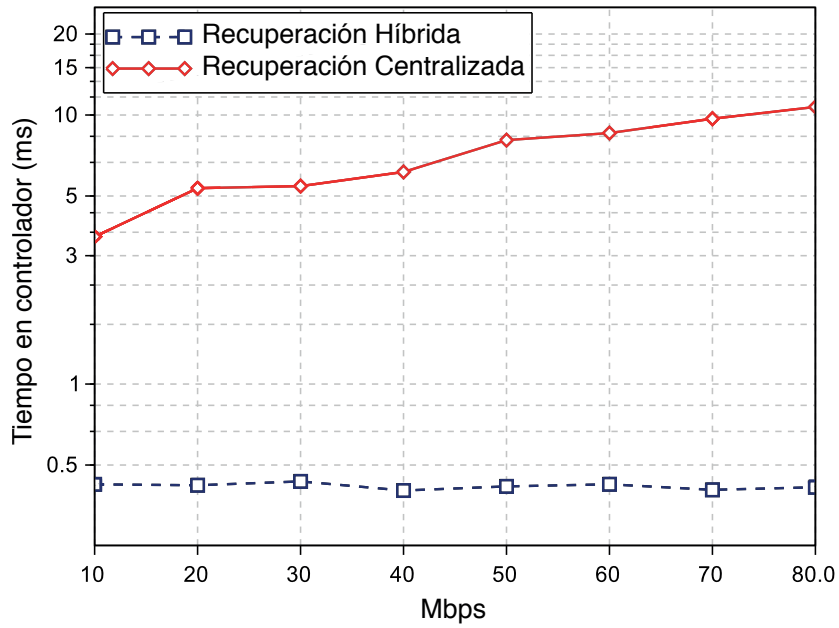


Figura 4.14: Comparación entre el procesamiento necesario en controlador obtenidos con distintos mecanismos [11]

4.5.3 Tiempo de recuperación en el controlador SDN

Por último, aprovechando las pruebas realizadas en las subsecciones anteriores se ha medido el tiempo que necesita el controlador para llevar acabo cada uno de los mecanismos de recuperación. Estos datos dan una visión del impacto que tienen en el controlador los diferentes procesos de recuperación.

Los resultados obtenidos tras estas mediciones se muestran en la figura 4.14 [11]. Lo primero que se observa en ficha figura es la existencia de dos gráficas únicamente. Este hecho se debe a que el mecanismo de *Recuperación Distribuida* no hace uso alguno del controlador y, por lo tanto, el tiempo del que hace uso del controlador es cero, esto hace que dicha recuperación no sea relevante para los resultados de esta prueba. Para el resto de los casos, se observa cómo el tiempo en que se hace uso del controlador por parte de la *Recuperación Híbrida* es muy inferior al tiempo utilizado por la *Recuperación Centralizada*. Esto se debe a que la *Recuperación Híbrida* necesita realizar unas tareas de muy bajo coste computacional, ya que solo debe localizar al switch destino, crear una nueva trama *Path Recovery* y encapsularla en un *Packet_Out* para su envío. Por su parte, la *Recuperación Centralizada* debe recalcular el nuevo camino, crear todos los *Mod_Flows* necesarios para modificar las reglas de reenvío y, por último, enviarlos a cada uno de los switches del nuevo camino para poder restaurar la comunicación.

4.6 Conclusiones

En este capítulo en primer lugar se ha presentado una comparación en términos de BW y FCT entre los switches *software* SDN más relevantes en la actualidad, OvS y BOFUSS. En esta comparación ha quedado patente la diferencia de rendimiento existente entre dichos dispositivos.

A continuación, se ha presentado un nuevo mecanismo de recuperación que aprovecha las bondades de los sistemas tradicionales (distribuidos), junto a las principales características de los nuevos paradigmas de red basados en SDN. Este mecanismo utiliza el conocimiento que tienen los controladores SDN junto a la versatilidad de los sistemas distribuidos para restablecer la comunicación entre dos elementos de la red.

Para la evaluación del mecanismo de recuperación se han realizado tres grupos de medidas mediante la realización de dos conjuntos de pruebas. El primer conjunto de pruebas ha comprobado el correcto funcionamiento del proceso de recuperación propuesto, mientras que el segundo conjunto ha evaluado el rendimiento de dicho proceso.

Centrando la visión en los resultados arrojados tras el primer grupo de medidas, donde se ha contabilizado el número de paquetes perdidos después de la aparición de un fallo en la red, se observa cómo el número de paquetes perdidos es muy pequeño en relación con el tamaño del flujo enviado. También se observa como el número de paquetes aumenta según aumenta la tasa de envío. Este aumento está dentro de lo esperado, puesto que, al aumentar la velocidad de transmisión de un flujo, el número de paquetes por unidad de tiempo aumenta y, por lo tanto, el número de paquetes perdidos dentro de ese intervalo también lo hace. Segundo conjunto de pruebas, donde el objetivo era medir cuánto de bueno es el proceso respecto a los otros dos mecanismos de recuperación (uno basado en computación centralizada mientras que el otro basado en una doble exploración), se puede concluir lo siguiente. El mecanismo propuesto tiene un buen rendimiento cuando la red tiene una carga media-baja ($\leq 70\,\%$) ya que el tiempo necesario para llevarla a acabo está por debajo de la recomendación establecida en [87], donde se recomienda que una recuperación no debe tardar más de 50 ms en restablecer la comunicación. Por otro lado, se observa como el tiempo que necesita el mecanismo propuesto es muy similar al tiempo utilizado por el mecanismo totalmente distribuido en cargas medias-bajas ($\leq 70\,\%$) y, a su vez, muy inferior al utilizado por el mecanismo centralizado. En los resultados, también se observa cómo al usar el nuevo mecanismo se produce una reducción del uso del controlador, lo que permite que este elemento clave de la arquitectura SDN pueda ser usada para otras labores de mayor prioridad.

Capítulo 5

Encaminamiento de flujos TCP mediante exploración de caminos

Las redes actuales ven como una infinidad de flujos pasan a través de ellas de manera diaria. Este uso constante provoca que las redes deban hacer uso de mecanismos, protocolos u algoritmos que les permitan optimizar sus características tales como uso de recursos, ancho de banda, fiabilidad, etc.

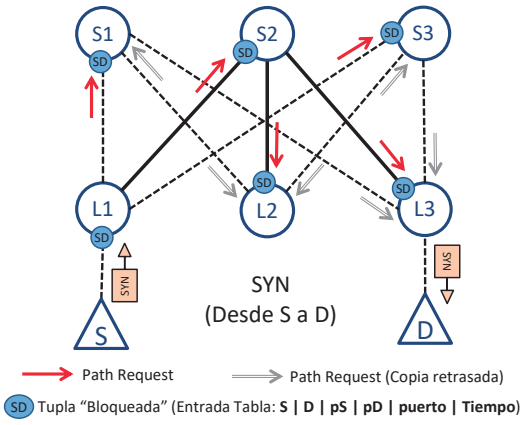
En el capítulo 3 se ha presentado un resumen de algunos protocolos y algoritmos diseñados para el establecimiento de caminos, ya sea por computación, exploración o simplemente configuración, que existen. El estudio, creación e implementación de estos sistemas se debe a la necesidad que tienen las redes para el establecimiento de caminos por el que encaminar un flujo entre un par de switches. Si se centra este estudio en las redes tradicionales, se observa como dichos protocolos y algoritmos están obligados a desplegarse de una manera distribuida. Asimismo, también han quedado reflejadas las deficiencias existentes en las diferentes propuestas existentes, entre ellas se quiere destacar la alta probabilidad de colisión entre flujos de gran tamaño cuya consecuencia suele ser una bajada drástica de rendimiento o un deficiente uso de los recursos de red lo cual generalmente viene causado por un mal reparto de la carga.

Tomando como punto de partida los problemas citados en el párrafo anterior (bajo rendimiento, deficiente uso de los recursos ofrecidos y mal balanceo de la carga), y observando que un altísimo porcentaje del tráfico que transita por las redes actuales es tráfico TCP [26], en esta tesis se propone un nuevo protocolo de encaminamiento en la capa de enlace diseñado para encaminar el tráfico TCP de forma eficiente, denominado *TCP-Path* [12], junto a una variante del mismo diseñada especialmente para aumentar su escalabilidad, denominada *TCP-Path for Elephants (TFE)*. Ambos mecanismos usan la exploración de la red mediante difusión para el establecimiento de los caminos, así como para obtener un mejor reparto de la carga dentro de la red y, por lo tanto, un mejor uso de sus recursos. TFE aplica el mecanismo de TCP-Path sólo a un subconjunto de los flujos TCP de la red.

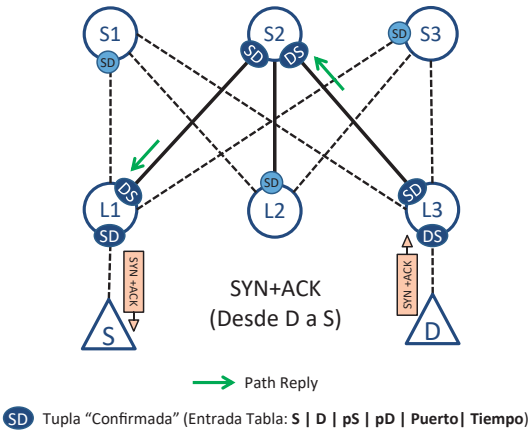
5.1 TCP-Path

TCP-Path [12] se define como un protocolo basado en la exploración de la red para el establecimiento del camino por donde se enviarán las tramas de un flujo TCP. Para entender el funcionamiento de este protocolo se va a hacer uso del ejemplo representado en la Figura 5.1 [12]. Se trata de una red jerárquica del tipo Spine-Leaf [85] , en la que se pretende establecer un flujo entre los equipos *S* y *D*. Siguiendo la definición propia del protocolo TCP, el equipo *S* comienza la fase de establecimiento de la comunicación TCP con la trama *TCP-SYN* de dicho protocolo.

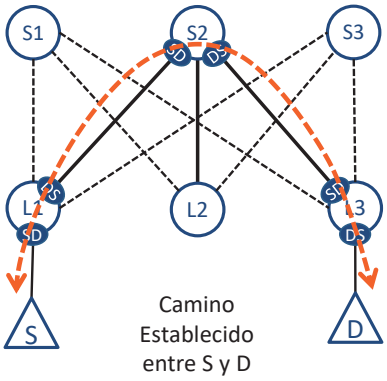
En el momento que el switch frontera *L1* recibe dicha trama, comienza la *fase de exploración de caminos*. Para ello, el switch *L1* encapsula la trama en una trama propia del protocolo TCP-Path denominada *Path Request*. Dicha trama sigue la estructura representada por la figura 5.2. Esta trama está compuesta por una dirección *Broadcast o Multicast* como *dirección MAC destino*, la dirección MAC origen del mensaje *TCP-SYN* original como *dirección MAC origen*, como *EtherType* el asignado para el *protocolo*



(a) Fase de exploración de caminos



(b) Fase de confirmación del camino



(c) Camino establecido

Figura 5.1: Fases del protocolo TCP-Path [12]

dirección MAC destino	dirección MAC origen	EtherType	Tipo de trama	Equipo destino
FF:FF:FF:FF:FF:FF	MAC equipo S	EtherType de TCP-Path	Path Request	MAC equipo D

Figura 5.2: Estructura y contenido de la trama *Path Request*

TCP-Path, *Path Request* como tipo de trama propia del protocolo y, por último, se inserta *la dirección MAC del equipo destino* del mensaje *TCP-SYN* original. Finalmente, el switch *L1* difunde la nueva trama reenviándola por todos sus puertos menos por el de entrada de la trama *TCP-SYN* original. Esta difusión comienza la creación de un árbol de confluencia con raíz en el switch frontera *L1* conectado directamente al equipo final *S* que inicia la comunicación.

Para evitar posibles bucles, este protocolo, implementa el mecanismo de prevención de bucles definido en el protocolo ARP-Path [44] y explicado en el capítulo anterior. Finalmente, el switch difunde la trama por todos sus puertos menos por el de entrada (ver figura 5.1a). Cabe citar también que en dicha figura se ha representado el momento de llegada de las tramas, es decir, las tramas representadas por las flechas rojas llegan a cada uno de los switches primero, lo que provoca el bloqueo del puerto de entrada, mientras que las tramas representadas por las flechas grises llegan más tarde y son rechazadas por cada uno de ellos.

Por último, cuando la primera trama *Path Request* llega al switch destino *L3*, esté la desencapsula, obteniendo de nuevo el mensaje *TCP-SYN* original. Finalmente, el switch reenvía el mensaje *TCP-SYN* obtenido al equipo destino, completando así la fase de exploración del camino.

La segunda fase del protocolo TCP-Path, denominada *Fase de confirmación de camino*, se inicia cuando el switch frontera destino recibe la trama *TCP-SYN+ACK*, que envía el equipo destino de la comunicación como confirmación de establecimiento del canal TCP. Este switch, al igual que su homónimo realizó en la primera fase, encapsula el mensaje en una trama propia del protocolo TCP-Path, denominada *Path Reply* (ver figura 5.3). La estructura de la trama *Path Reply* está compuesta por la *dirección MAC destino* del mensaje *TCP-SYN+ACK* como *dirección MAC destino* (en este caso MAC equipo S), la *dirección MAC origen* del mensaje *SYN+ACK* como *dirección MAC origen* (en este ejemplo MAC Equipo D), como *EtherType* el asignado para el protocolo TCP-Path y, por último, *Path Reply* como tipo de trama propia del protocolo.

Una vez confirmado el camino, cada uno de los conmutadores que conforman dicha ruta poseen una entrada en sus tablas de encaminamiento, como la representada por la tabla 5.1, gracias a la cual son capaces de encaminar las tramas pertenecientes al flujo establecido entre el equipo *S* y el equipo *D* (ver figura 5.1c).

Dirección MAC destino	Dirección MAC origen	EtherType	Tipo de trama
MAC equipo D	MAC equipo S	EtherType de TCP-Path	Path Replay

Figura 5.3: Estructura y contenido de la trama *Path Reply*

Dirección MAC destino	Dirección MAC origen	Puerto TCP origen	Puerto TCP destino	Puerto salida	Timer
Dirección MAC equipo S	Dirección MAC equipo D	Puerto origen	Puerto destino	Puerto	Timer

Tabla 5.1: Estructura y contenido entrada en tabla TCP-Path

La regla almacenada en cada uno de los switches está compuesta por, primero, los campos encargados de definir de forma unívoca un flujo: dirección MAC origen del flujo, dirección MAC destino de flujo, Puerto TCP origen, Puerto TCP destino, el puerto de salida para poder encaminar el flujo al siguiente switch y, por último, una marca temporal o *timestamp* para establecer la posible caducidad de la regla.

5.1.1 Diseño Funcional TCP-Path

Una vez definido el mecanismo propuesto por el protocolo TCP-Path para explorar y confirmar los caminos se pasa a explicar el diseño funcional del protocolo. Para esta explicación se va a hacer uso de la figura 5.4.

En el momento que una trama es recibida por uno de los switches de la red, el switch comprueba si el *EtherType* de la trama coincide con el código asignado al protocolo

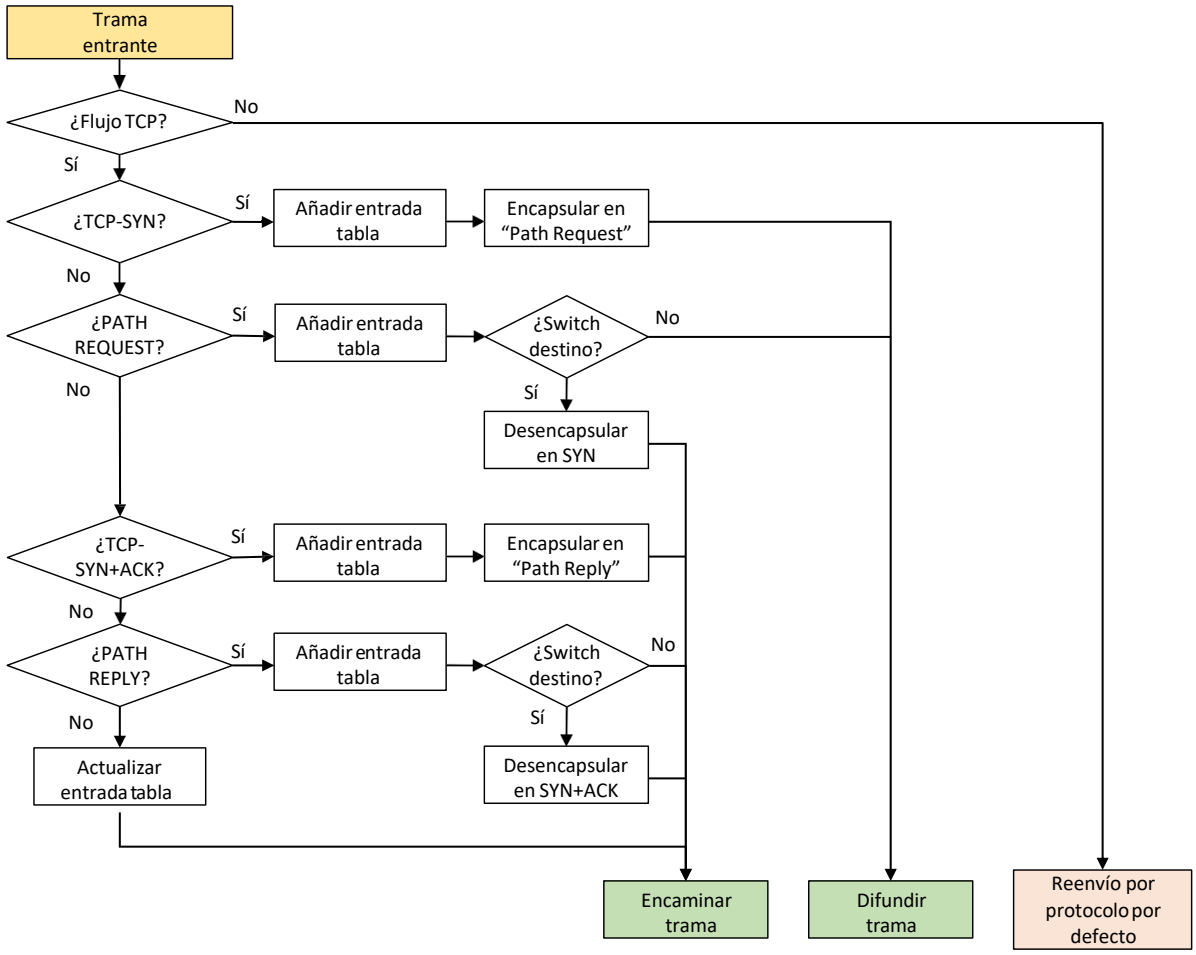


Figura 5.4: Diagrama de flujo TCP-Path

TCP-Path, o si la cabecera IP de la trama tiene '6' como valor del campo *protocolo* [88], comprobando de esta manera que la trama tiene como protocolo de transporte a TCP. Si la verificación es correcta, se procede a aplicar las diferentes acciones definidas por TCP-Path, en caso contrario, será el protocolo de encaminamiento por defecto el que se encargue de reenviar dicha trama.

Las acciones del protocolo TCP-Path se pueden agrupar según el tipo de la trama entrante. En el caso de que la trama de entrada sea una trama TCP se pueden diferenciar tres casos diferentes:

- Si la trama recibida es del tipo *TCP-SYN*, el sistema añade una nueva entrada en la tabla de encaminamiento (ver tabla 5.1) y, posteriormente, encapsula la trama *TCP-SYN* en una trama *Path Request* (ver figura 5.2). Por último, difunde la nueva trama por todos sus puertos menos por el de entrada, comenzando de este modo el proceso de exploración.
- Si la trama recibida es un *TCP-SYN+ACK*, el switch añade una nueva entrada en la tabla de encaminamiento (siguiendo la estructura representada en la tabla 5.1) y encapsula el mensaje en una trama *Path Reply* (ver figure 5.3). Por último, reenvía dicha trama a través del puerto guardado en la fase de exploración dentro de la tabla de encaminamiento.
- En última instancia, si la trama TCP no está dentro de los casos anteriores, encamina dicha trama a través del puerto almacenado en la tabla de encaminamiento. Dicho puerto se encuentra en el registro almacenado en dicha tabla y cuyos parámetros (direcciones MAC origen, direcciones MAC destino y puertos TCP origen y destino) coincidan con los establecidos en la cabera de la trama entrante.

Por el contrario, si la trama entrante es propia del protocolo TCP-Path se tienen únicamente dos posibles casos:

- En el caso de recibir una trama *Path Request* (ver figura 5.2), el switch comienza por aplicar el mecanismo de prevención de bucles explicado anteriormente, es decir, el switch comprueba que no exista una entrada previa en su tabla de encaminamiento (ver tabla 5.1). De ser así, automáticamente descarta la trama para evitar bucles. En caso contrario (no existe ningún registro), el switch crea un nuevo registro en la tabla y comprueba si él es el switch destino frontera del flujo. En caso afirmativo, dado que el switch desencapsula la trama *Path Request* (ver figura 5.2) para obtener el mensaje *TCP-SYN* original y enviárselo al equipo destino del flujo. En caso contrario, el switch que recibe la trama no es el switch destino del flujo, simplemente difunde la trama por todos sus puertos menos por el de entrada.
- Si la trama recibida es del tipo *Path Reply* (ver figura 5.3), el switch que recibe la trama inserta una nueva entrada en la tabla de encaminamiento (ver tabla 5.1),

busca en dicha tabla el puerto de salida para reenviar la trama y, por último, comprueba, al igual que en el caso anterior, si él es el switch frontera destino. De ser así, obtendrá el mensaje original (*TCP-SYN+ACK*) desencapsulando la trama *Path Reply* para, posteriormente, enviarlo al equipo final destino del flujo a través de dicho puerto. En caso contrario, si el switch no es el switch frontera destino, únicamente reenvía la trama *Path Reply* (ver figura 5.3) por el puerto encontrado en la tabla de encaminamiento. Para reducir la computación necesaria en los switches, este paso puede ser simplificado no realizando el encapsulado del mensaje *TCP-SYN+ACK* en una trama *Path Reply*.

5.1.2 Mecanismo anti-rebote en el proceso de difusión

El protocolo presentado, TCP-Path, basa su generación de caminos en la exploración mediante difusión de tramas por toda la red. Esta propagación utiliza el mecanismo de bloqueo definido en el protocolo ARP-Path [44] para evitar los bucles. Sin embargo, no elimina la posibilidad de crear caminos subóptimos cuyo número de saltos es mayor que el mínimo existente entre dos equipos finales de la red(ver figura 5.5), algo nada deseable ya que estos saltos pueden provocar que haya un uso inadecuado de recursos para un único camino, o lo que es peor, si el flujo que toma dicho camino es un flujo de tipo *Elefante* o lo que es lo mismo un flujo de gran tamaño, puede generar una caída general del rendimiento debido al uso intensivo de todos los enlaces del camino, generando un cuello de botella para el resto de flujos de la red.

Para solucionar este problema se presenta un mecanismo *anti-rebote* capaz de prevenir la generación de caminos con múltiples saltos entre capas en redes jerárquicas. Este mecanismo necesita que cada uno de los switches que conforman la red conozca en que capa de la misma se encuentra situado. Para que cada switch conozca la capa donde está

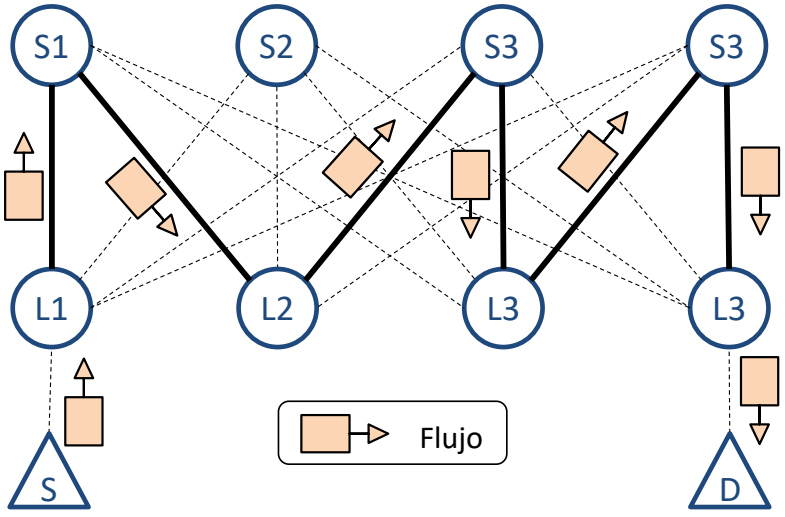


Figura 5.5: Camino generado sin mecanismo anti-rebote

situado se da un nuevo uso al paquete *Hello*. Este paquete *Hello* es una trama de nivel 2 que el switch envía por todos sus puertos (excepto, si existiera, el puerto conectado con el controlador), está formada por una cabecera *Ethernet* compuesta por: una *dirección MAC multicast* como *dirección MAC destino*, la dirección MAC del primer puerto del conmutador y un *EtherType* específico. Al ser una trama que va a ser usada en redes *Ethernet* debe tener una longitud mínima, por ello se ha asociado también un *payload* en el cual los switches insertarán, en caso de conocerlo, el valor de la capa de la red donde están situados, para ello usarán uno de los siguientes valores como indicador de capa: '1' para la capa situada en la parte más 'alta' de la red o lo que es lo mismo la capa *CORE*, '2' para la capa situada en el medio de la red o lo que es lo mismo la capa de *Agregación*, '3' para la capa conectada directamente a los equipos finales o también llamada capa de *ToR* y '0' cuando desconozcan la capa en la que se sitúan.

Para poder insertar el valor asignado a la capa en la que se sitúa cada uno de los switches en los paquetes *Hellos* se requiere un conocimiento previo de dicho valor por parte del switch. Para obtener esta información se presenta el algoritmo 5.1.

Dicho algoritmo 5.1 necesita que los switches situados en la capa más alta de la topología, *Cores*, sean configurados explícitamente por los operadores de red. Esta información previa permitirá a los switches aprender de una forma distribuida su posición en la red. Del mismo modo, los switches situados en la capa intermedia de la red, *Agregados*, conocerán su posición debido a que recibirán paquetes *Hello* por todos sus puertos, ya que están conectados con los switches *Cores* por un lado y con los switches *ToRs* por otro. Por último, los switches situados en la capa más baja de la red, *ToRs*, conocerán su posición ya que solo recibirán paquetes *Hello* por los puertos conectados a los switches *Agregados* (los paquetes *Hellos* son emitidos por los switches de la red y no por los equipos finales). Por supuesto, puede existir un tiempo transitorio donde los equipos aún no conozcan la posición exacta donde se sitúan, no obstante, se considera despreciable.

Para que el switch pueda identificar si el puerto por el que ha recibido una trama está conectado con un switch situado en la capa *CORE* o con un switch posicionado en la capa *Agregada* o, por el contrario, si la trama proviene de un equipo final directamente, el switch

```
Data: Paquete entrante
Result: Paquete Hello con información de situación
begin
    if Capa desconocida then
        Enviar paquete Hello con valor 0;
    else if Configuración Core == True then
        El switch es un Core; Enviar paquete Hello con valor 1;
    else if Todos los puertos han recibido paquete Hello then
        El switch es un Agregado; Enviar paquete Hello con valor 2;
    else
        El switch es un ToR; Enviar paquete Hello con valor 3;
```

Algoritmo 5.1: Inserción del valor de la capa en la que se sitúa el switch en los paquetes *Hello*

```
Data: Trama entrante
Result: Difusión de tramas Broadcast
begin
    if switch Core == True then
        Difusión por todos los puertos;
    if switch Agregado == True then
        if Puerto de entrada == Puerto conectado con CORE then
            Difusión solo por puertos conectados con ToRs
        else if Puerto de entrada == Puerto conectado con ToR then
            Difusión por todos los puertos;
    else
        if Puerto de entrada == Puerto conectado con Agregado then
            Difusión solo por puertos conectados con Equipos Finales
        else if Puerto de entrada == Puerto conectado con Equipos Finales then
            Difusión por todos los puertos;
```

Algoritmo 5.2: Proceso de difusión con anti-rebote

dispone de una tabla donde almacena la información de sus vecinos. Los registros están estructurados de la siguiente manera: *dirección MAC*, *puerto de entrada*, *código de posición en la red*. Esta tabla se rellena gracias a la recepción de los mensajes *Hellos*, que, como es evidente, solo puede contener información de los switches vecinos, pero no de equipos finales. Por lo que, para identificar cuando un puerto está conectado directamente con un equipo final, el propio switch debe comprobar que no existe ningún registro asociado a dicho puerto dentro de su *tabla de vecinos*.

Una vez los switches conocen la posición que ocupan dentro de la red, el mecanismo establece (ver algoritmo 5.2) que los switches identificados como *Cores* difundirán las tramas por todos sus puertos siempre que el sistema de bloqueo anti-bucles lo permita. Los switches identificados como *Agregados* difundirán las tramas por todos los puertos si la trama original entró por uno de los puertos conectados con los switches situados en la capa *ToR*, si, por el contrario, la trama ha entrado por un puerto conectado con un switch situado en la capa *CORE*, el switch solo difundirá la trama por los puertos conectados con los equipos situados en la capa *ToR*. Por último, si el switch se encuentra situado en la capa inferior de la red (capa *ToR*), difundirá por todos sus puertos si la trama original entró por un puerto conectado con un equipo final, en caso contrario solo difunde por aquellos puertos conectados directamente con los equipos finales.

5.1.3 Implementación de TCP-Path en un Switch Software

Para llevar a cabo la implementación del protocolo, se ha tomado como punto de partida también la implementación realizada en AOSS [43]. Para la realización de la implementación se ha de introducir una modificación funcional en el esquema de funcionamiento establecido en dicho conmutador. Esta modificación se sitúa por debajo de la búsqueda establecida por el protocolo OpenFlow dentro de la *Miss-table* e inmediatamente por encima de la integración del protocolo ARP-Path realizada en AOSS. La modificación queda representada en la figura 5.6.

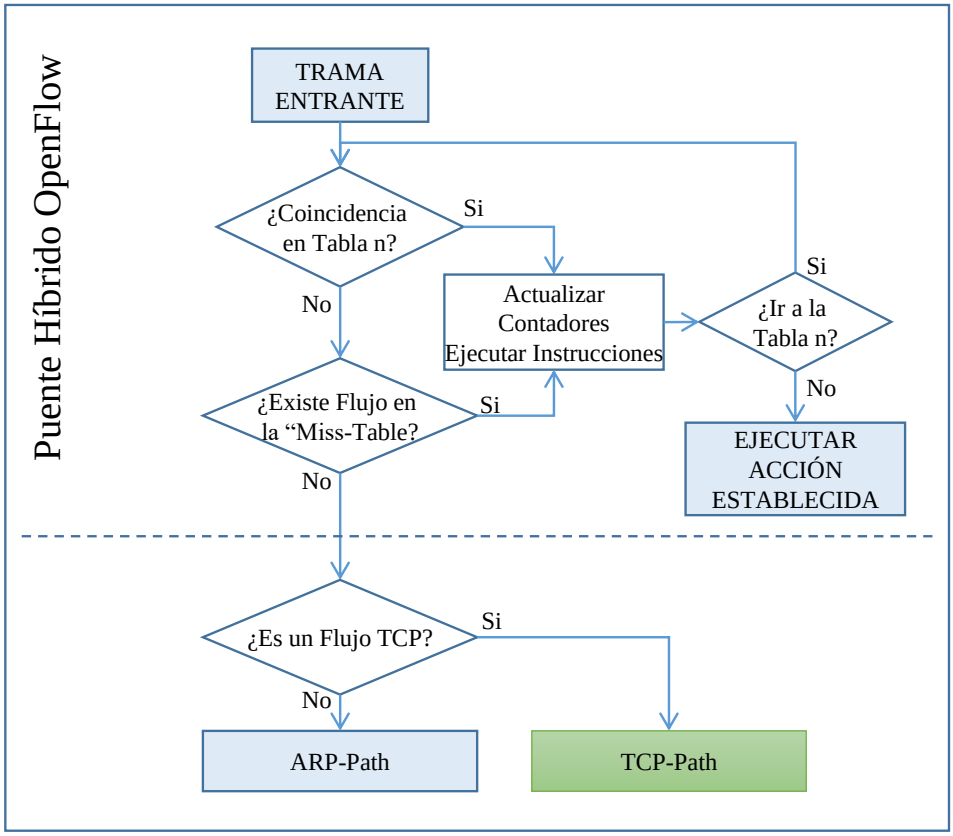


Figura 5.6: Diagrama de flujo switch TAOSS

Este desarrollo se realiza mediante la implementación de dos funciones diferentes, una primera función donde se comprueba el tipo de trama recibida, es decir, si la trama debe ser tratada por el protocolo TCP-Path o, por el contrario, debe ser tratada por el protocolo de encaminamiento por defecto. Esta primera función realiza una doble búsqueda dentro de la trama entrante, la primera comprueba si el valor del campo *EtherType* de la cabecera *Ethernet* coincide con el valor asignado al protocolo TCP-Path o si, por el contrario, teniendo un valor, '*0x0800*' para el campo *EtherType* y '6' como valor del campo *protocolo* dentro de la cabecera *IP* o, si es una trama específica del protocolo TCP-Path. Esta lógica queda resumida en el algoritmo 5.3.

5.1.3.1 Procesamiento general de tramas en TCP-Path

La segunda función diseñada e implementada para llevar a cabo el proceso de encaminamiento y exploración de caminos descrito en el protocolo propuesto es el *Procesamiento general de tramas*. Esta función es la encargada de establecer que tratamiento va a recibir cada uno de las tramas que han sido detectadas, bien como flujos TCP, bien como propias del protocolo TCP-Path por la algoritmo 5.3. Como se puede ver en el algoritmo 5.4, la función de *Procesamiento general de tramas en TCP-Path* realiza una primera división según el tipo de trama recibida. Para el caso de una trama con cabecera TCP, esta función a su vez establece tres posibles casos para su tratamiento: el

```
Data: Cabecera de la trama
Result: Siguiente manejador de la trama entrante
begin
    if Trama->Ethernet->EtherType == '0x0800' and Trama->Ip->Protocolo == '6' then
        if Trama->TCP != NULL then
            Encaminar por protocolo TCP-Path;
        else
            Eliminar trama mal formada;
    else if Trama->Ethernet->EtherType == 'TCP-Path EtherType' then
        if Trama->TCP-Path != NULL then
            Encaminar por protocolo TCP-Path;
        else
            Eliminar trama mal formada;
    else
        Encaminar por protocolo de encaminamiento por defecto;
```

Algoritmo 5.3: Selector de la siguiente función que debe manejar la trama

```
Data: Cabecera Trama TCP
Result: Siguiente manejador de trama TCP
begin
    if Trama->TCP != NULL then
        if Trama->TCP->Flags == "SYN+ACK" then
            Encapsular en "Path Reply";
        else if Trama->TCP->Flags == "SYN" then
            Encapsular en "Path Request";
        else
            reenviar trama Ünicast";
    else if  Trama->TCP-PATH != NULL then
        if Trama->TCP-PATH->Tipo == Request" then
            Tratar trama "Path Request";
        else if Trama->TCP-PATH->Tipo == Reply" then
            Tratar trama "Path Reply";
        else
            Descartar trama errónea;
    else
        Encaminar por protocolo de encaminamiento por defecto;
```

Algoritmo 5.4: Procesamiento general de las tramas en TCP-Path

primero, en el caso de tratarse de una trama *TCP-SYN*; el segundo para el caso de tratarse de una trama *TCP-SYN+ACK*, por último, para cualquier otro caso. Del mismo modo, esta función también establece el tratamiento de las tramas propias del protocolo y, al igual que en el caso anterior, realiza una división según la trama recibida. Por seguridad, ante la posibilidad de fallo de la función 5.3, esta función establece un tercer caso para todas aquellas tramas que no cumplan con los requisitos de los dos casos anteriores. En este tercer caso, esta función reenvía la trama a la función diseñada por el protocolo por defecto, en el caso de esta tesis, el protocolo ARP-Path.

5.1.3.2 Tratamiento de tramas TCP: SYN/SYN+ACK

Las tramas *TCP* que tengan el bit *SYN* o los bits *SYN* y *ACK* activos, como indica el algoritmo 5.4, deben ser tratadas de forma especial. Por ello, se diseña e implementa

una función para su encapsulado. Esta función es la encargada de, por un lado, obtener los datos de la cabecera TCP de dichas tramas y, por otro lado, generar una de las dos nuevas tramas propias del protocolo propuesto. Para el caso de una trama donde solo se encuentre activo el bit *SYN*, el sistema genera un *Path Request*, mientras que si el sistema detecta que los bits *SYN* y *ACK* se encuentran activos se procede a crear una trama *Path Reply*.

Para su correcta implementación es necesario, en primer lugar, introducir una nueva estructura de datos representada por el código 5.1. Como se puede observar en ella, se trata de una copia ampliada de la estructura definida para las tramas TCP. Esta extensión se debe a que el protocolo TCP-Path necesita dos campos más para poder realizar correctamente la encapsulación o desencapsulación de las tramas: el primer campo es el *Option Code* (*Op*), el cual indica el tipo de trama TCP-Path que se genera (si *Op* tiene como valor **1** se trata de un *Path Request*, mientras que si el valor de *Op* es **2** se trata de un *Path Reply*); el segundo campo es la dirección MAC destino del flujo. Esto se debe a que en el caso de la trama *Path Request* convertimos la trama original *unicast* en una trama de difusión y, por lo tanto, se debe cambiar la dirección MAC destino de la cabecera Ethernet por una dirección MAC de difusión; en el caso de la implementación se ha usado la dirección de broadcast (*FF:FF:FF:FF:FF:FF*). Además, hay que almacenar la dirección MAC destino original en otro campo de la nueva trama, para luego reconstruir la trama original y hacérsela llegar al destinatario.

Código 5.1: Definición de la estructura de las tramas TCP-Path

```
#define PATH_OP_REQUEST 1
#define PATH_OP_REPLY 2
#define PATH_OP_FLUSH 3
#define PATH_ETH_HEADER_LEN 34

//nueva estructura para paquetes TCP-Path
struct path_header {
    uint16_t tcp_src;
    uint16_t tcp_dst;
    uint32_t tcp_seq;
    uint32_t tcp_ack;
    uint16_t tcp_ctl;
    uint16_t tcp_winsz;
    uint16_t tcp_csum;
    uint16_t tcp_urg;
        /* Option code. */
    uint8_t op;
        /* Sender hardware address. */
    uint8_t mac_dst[ETH_ADDR_LEN];
        /* id switch. */
        uint16_t id_dp;
        /*Campos para checkear desarrollo */
        /* Secuencia to track. */
    uint32_t secuence;
        /* Contador de salto */
        uint8_t contador;

} __attribute__((packed));

BUILD_ASSERT_DECL(PATH_ETH_HEADER_LEN == sizeof(struct path_header));
```

Código 5.2: Nueva función de parseo dentro de la lógica de NebBee

```
if (protocol_Name.compare("path") == 0 && pkt_proto->path == NULL)
{
        pkt_proto->path = (struct path_header *) ((uint8_t*) pktin->data + proto->Position);
        PDMLReader->GetPDMLField(proto->Name, (char*) "sport", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_TCP_SRC);
        PDMLReader->GetPDMLField(proto->Name, (char*) "dport", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_TCP_DST);
        PDMLReader->GetPDMLField(proto->Name, (char*) "op", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_OP); //opcion
        PDMLReader->GetPDMLField(proto->Name, (char*) "MacDst", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_MAC_DST); //mac dst
        PDMLReader->GetPDMLField(proto->Name, (char*) "id_dp", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_ID_SWITCH); //id  switch
        PDMLReader->GetPDMLField(proto->Name, (char*) "secuence", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_SECUENCE); //secuence
        PDMLReader->GetPDMLField(proto->Name, (char*) "contador", proto->FirstField, &field);
        nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_PATH_SALTOS); //Saltos
}
```

Del mismo modo, es necesario realizar un cambio adicional en los archivos originales del *OfSoftSwitch13* [40]. Esta vez el cambio se realiza en la librería *NetBee*. Esta librería es la encargada de extraer los datos de las cabeceras de las tramas y, también, de transformar los datos de las estructuras del switch en tramas. El cambio introduce una estructura '*if*' (código 5.2) que le indica a la librería primero cómo debe serializar los nuevos datos, es decir, en qué posición de la trama se sitúan y cómo debe deserializar los datos procedentes de las distintas cabeceras que formas la trama para introducirlos correctamente en las nuevas estructuras creadas en el código 5.1. El segundo cambio indica al sistema la existencia de una nueva cabecera, correspondiente a las tramas especiales del protocolo, así como el tipo de datos que contiene, cómo se organizan dentro de la cabecera, el tamaño de cada uno de ellos y su valor (código 5.3).

Por último, es necesario diseñar e implementar la lógica encargada de generar estas nuevas tramas a partir de las tramas TCP originales. Para ello, se define y establece la función *Encapsulate Path TCP*, cuya lógica se ha representado en el algoritmo 5.5. Dicho código se encarga de completar las estructuras de datos pertenecientes a TCP-Path copiando en ella los valores de los campos de la estructura TCP. Posteriormente copia en el campo *MAC Destino* de la cabecera TCP-Path el valor de la dirección MAC destino de la cabecera *Ethernet* de la trama original y, dependiendo de los bits TCP activos, establece diferentes acciones. En el caso de encontrar solamente el bit *SYN* activo, se introduce '1' (*Path Request*) como valor del campo *Op* de la cabecera TCP-Path y se fija

Código 5.3: Nueva casuística CustomNetPdl

```
<protocol name="path" longname="Tcp-path Pkt" comment="Tcp-path" showsumtemplate="Tcp-path">
        <format>
                <fields>
                        <field type="fixed" name="sport" longname="{0x8000 43} sport"size="2" showtemplate="FieldDec"/>
                        <field type="fixed" name="dport" longname="{0x8000 44} dport" size="2" showtemplate="FieldDec"/>
                        <field type="bit" name="flags" longname="Flags" mask="0x003F" size="8" showtemplate="FieldHex">
                        <field type="fixed" name="op" longname="{0x8000 40} op" size="1" showtemplate="path.op"/>
                        <field type="fixed" name="MacDst" longname="{0x8000 41} MacDst" size="6" showtemplate="MACaddressEth"/>
                </fields>
        </format>
</protocol>
```

```
Data: Trama TCP SYN/SYN+ACK
Result: Trama Path Request/Reply
begin
    Datos Estructuras TCP-Path = Datos Estructuras TCP;
    Trama->TCP-Path->Mac_Destino = Trama->Ethernet->Mac_Destino;
    if Trama->TCP->Flags == "SYN" then
        TCP-Path->Op = 1;
        Trama->Ethernet->Mac_Destino = "FF:FF:FF:FF:FF:FF";
    else if Trama->TCP->Flags == "SYN+ACK" then
        TCP-Path->Op = 2;
    else
        Descartar trama errónea;
        Return "NULL";
    if Validación de la trama con NBee == OK then
        Return "Trama";
    else
        Return "NULL";
```

Algoritmo 5.5: Encapsulado de las tramas TCP-Path

la dirección MAC de difusión como dirección MAC destino de la cabecera *Ethernet*. Para el caso de encontrar tanto el bit *SYN* como el bit *ACK* activos, únicamente fija el valor del campo *Op* de TCP-Path al valor '2' (*Path Reply*). Finalmente, realiza una validación de la trama para comprobar que la creación de la misma ha sido correcta.

5.1.3.3 Tratamiento de las tramas especiales Path Request y Path Reply

Otro de los puntos importantes de este desarrollo es la manera de tratar las tramas especiales del protocolo. Estas tramas se enfrentan a dos tratamientos diferentes dependiendo del switch que las maneje. Si el switch que lo recibe es un switch que no está conectado con ningún equipo final directamente, denominado switch *interior*, entonces se verá manejado por la función cuya lógica se representa mediante el algoritmo 5.6. En este algoritmo se ve como cuando un switch interior recibe una trama propia del protocolo, lo primero que hace es determinar de qué tipo de trama propia se trata. En el caso de tratarse de una trama del tipo *Packet Request*, el switch busca en su tabla de encaminamiento si existe un registro para la trama entrante, en el caso de no existir ningún registro crea un nuevo registro donde inserta las direcciones MAC origen y destino, los puertos TCP origen y destino, el puerto de entrada de la trama y un *timestamp*. A continuación, difunde la trama por todos los puertos menos por él de entrada. Por el contrario, si el switch dispone de un registro para dicha trama, la descarta para evitar bucles. Del mismo modo, si la trama recibida es del tipo *Packet Reply* el switch procede a buscar en su tabla de salida el puerto por el que debe encaminar la trama. Si no dispone de dicho puerto inicia el proceso de recuperación de camino, en caso contrario, crea un nuevo registro con las direcciones MAC origen y destino, los puertos TCP origen y destino, el puerto de entrada de la trama y un *timestamp*. Actualiza el *timestamp* del registro de salida y, a continuación, encamina la trama por el puerto de salida. Por el contrario, si el switch

```
Data: Trama TCP-Path
Result: Encaminamiento de las tramas TCP-Path
begin
    /* Si la trama es una trama propia del protocolo                    */
    if Trama->Ethernet->EtherType == "TCP-Path EtherType" then
        if Trama->TCP-Path->Op == 1 then
            Puerto-Entrada-tabla = Buscar puerto de entrada en tabla();
            if Puerto-Entrada-tabla == NULL then
                Insertar una nueva entrada en tabla TCP-Path;
                Difundir la trama por todos los puertos menos por el de entrada;
            else
                Eliminar la trama para evitar bucles;
        else if Trama->TCP-Path->Op == 2 then
            Puerto-salida = Buscar puerto de salida en tabla();
            if Puerto-salida no Existe then
                Comenzamos la Recuperación;
            else
                Insertamos nueva entrada en tabla TCP-Path;
                Actualizar la marca temporal del registro;
                Encaminar la trama por puerto de salida;
```

Algoritmo 5.6: Manejador de las tramas propias del protocolo TCP-Path en switches no fronteras

que recibe la trama es un switch frontera, es decir, se encuentra conectado directamente con al menos un equipo final, entonces el manejador de tramas TCP-Path sigue la lógica representada en el algoritmo 5.7. Este algoritmo, a diferencia del algoritmo 5.6, necesita, por un lado, conocer la información de los equipos finales conectados directamente a él y, por otro lado, tener la capacidad de recuperar la trama original correspondiente haciendo un desencapsulado de la trama recibida. Para ello, el switch lo primero que realiza nada más recibir la trama es obtener la información de los equipos finales conectados a él, para posteriormente comprobar si el equipo final destino de la trama está entre ellos. En caso afirmativo, el switch recupera la trama original y se la reenvía al equipo final a través del puerto correspondiente. En caso contrario, el switch se comporta como un switch interior y aplica las mismas reglas de reenvío descritas en el algoritmo 5.6.

Como se ha descrito en el párrafo anterior, los switches conectados directamente con al menos un equipo final (switches frontera) necesitan de una función capaz de obtener las tramas originales TCP a partir de las tramas propias del protocolo TCP-Path. La función implementada para esta labor se muestra en el algoritmo 5.8. Esta función se inicia

```
Data: Trama de red (IP)
Result: Encaminamiento de la trama
begin
    Vecino = Buscar en tabla de equipos vecinos; if Vecino == TRUE and
     Trama->Ethernet->EtherType == "TCP-Path EtherType" then
        Desencapsular la trama TCP-Path para obtener la trama TCP Original;
        reenviar la trama por puerto conectado con equipo final;
    else
        Aplicar algoritmo 5.6
```

Algoritmo 5.7: Switch frontera TCP-Path

```
Data: Trama Path Request/Reply
Result: Trama TCP SYN/SYN+ACK
begin
    Datos Estructuras TCP = Datos Estructuras TCP-Path
    TCP-Path es una extensión de TCP;
    MAC_Destino_Ethernet = TCP-Path MAC Destino ;
    if TCP-Path->Op == 1 then
        Trama->TCP->Flags = "SYN";
    else if TCP-Path->Op = 2 then
        Trama->TCP->Flags = "SYN+ACK";
    else
        Descartar trama errónea
    Validar la trama con la librería NetBee;
```

Algoritmo 5.8: Desencapsulado de las tramas en TCP-Path

```
Data: Trama TCP
Result: Encaminamiento de las tramas TCP
begin
    Puerto_Salida = Buscar Puerto en tabla de encaminamiento de TCP-Path;
    if Puerto_Salida == NULL then
        Iniciar Recuperación;
    else
        Actualizar la marca temporal del registro;
        Encaminar la trama por puerto de salida;
```

Algoritmo 5.9: Encaminamiento de los mensajes TCP

creando una nueva estructura de datos TCP y completando sus valores con los valores contenidos en la estructura de datos de TCP-Path. A continuación se copia la dirección MAC destino almacenada en el campo *dirección MAC destino* de la cabecera TCP-Path. Del mismo modo, el switch, para seleccionar los flags activos de TCP comprueba el campo *Op* de TCP-Path, es decir si el valor es '1' el flag activo es *SYN* mientras que si el valor es '2' los flags activos son *SYN* y *ACK*. Para finalizar, el switch valida la nueva trama haciendo uso de la librería *NetBee*.

5.1.3.4 Tratamiento de los mensajes TCP

Por último, se ha diseñado e implementado una función para manejar todas aquellas tramas que no han sido tratadas anteriormente, es decir, una función que pueda manejar el resto de los mensajes TCP. Para generar esta función se ha diseñado la lógica mostrada en el algoritmo 5.9.

En este algoritmo se observa como el switch comienza por buscar en su tabla de encaminamiento TCP-Path el puerto por el que debe reenviar la trama entrante. Si no existe dicho puerto, inicia el mecanismo de recuperación de caminos que tenga configurado, en caso contrario, actualiza el *timestamp* del registro donde ha encontrado el puerto de salida para, a continuación, encaminar el mensaje TCP por dicho puerto.

Característica	Valor
Topología de la red	Spine-Leaf (4 - 4)[85]
Equipos por Leaf	20
Matriz de Tráfico	Aleatoria entre *Leaf*
Distribuciones tamaño de flujos	Web Search[83] & Data Mining [84]
Carga ofrecida a la Red (%)	10, 20 & 40 %
Velocidad de los enlaces	10 Mbps (emulación y simulación) & 100 Mbps (Simulación)
Tiempo de prueba (s)	1800 s
Tiempo transitorio (s)	800 s
Número de repeticiones	10

Tabla 5.2: Configuración para evaluación de TCP-Path

5.1.4 Evaluación

La evaluación del protocolo propuesto, TCP-Path, se ha realizado mediante una comparación en términos de BW y FCT entre TCP-Path y ECMP bajo diferentes cargas. Se ha elegido ECMP por ser actualmente el algoritmo de referencia para el encaminamiento de tramas en centros de datos.

Para la llevar a cabo estas pruebas se ha reutilizado la configuración de las pruebas realizadas en la subsección 4.1.1. En dichas pruebas se utiliza la topología *Spine-Leaf* compuesta por 4 *Spine*, 4 *Leaf* y 20 equipos por *Leaf*, junto a una matriz de tráfico aleatorio y dos distribuciones de tamaño de flujo diferentes (*Web Search*[83] y *Data Mining*[84]). Como resumen de las configuraciones establecidas para las pruebas se muestra la tabla 5.2.

Por último, para la realización de las pruebas se ha contado con una infraestructura formada por 7 ordenadores con procesador Intel(R) Core(TM) i7 y 24 GBytes de Memoria RAM, todos ellos interconectados mediante un switch *Netgear GS116switch*. Dicha infraestructura se ha utilizado usando dos tipos de herramientas diferentes: la herramienta de simulación Network Simulator 3 (ns-3) [89] en su versión ns-3.24 y la herramienta de emulación de redes Mininet [82]. El uso de dos plataformas distintas para el mismo banco de pruebas tiene como explicación la validación del modelo simulado en ns-3 mediante un modelo emulado en Mininet ya que, ns-3, al tratarse de un simulador, posee mayor escalabilidad que Mininet. Mientras que, Mininet permite realizar pruebas utilizando la implementación real del stack TCP/IP de Linux y Cubic como algoritmo de control de congestión, mientras que ns-3 utiliza una implementación propia del stack y Vegas como algoritmo de control de congestión.

5.1.5 Resultados

Una vez presentadas las pruebas que se han realizado para la evaluación del protocolo propuesto, en esta sección se analizan y presentan los resultados obtenidos tras la realización de dichas pruebas.

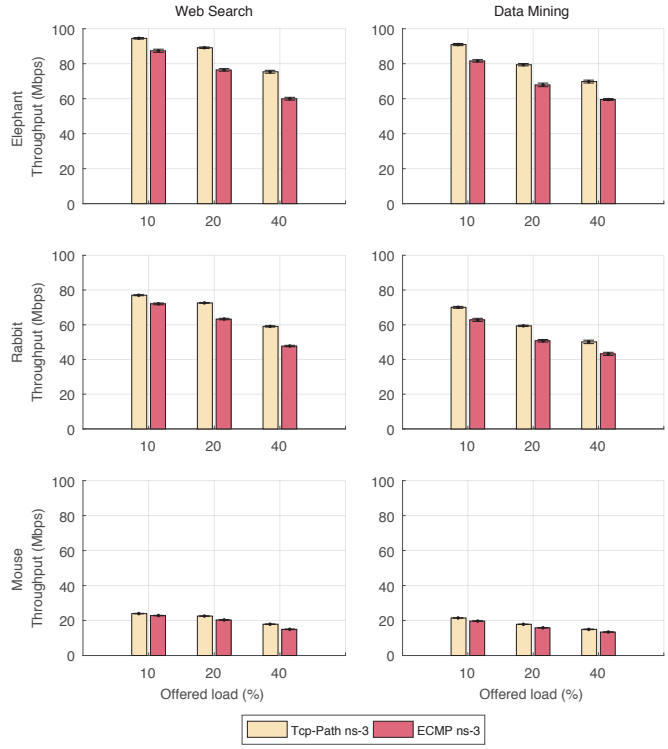
5.1.5.1 Rendimiento en términos de BW y FCT

Las pruebas ejecutadas en el simulador fijando la velocidad de transmisión de los enlaces 100 Mbps dieron como resultados los datos mostrados en la figura 5.7 [12]. La figura 5.7a representa el BW medio de los diferentes flujos, mientras que la figura 5.7b representa el FCT medio de los mismos. La gráficas representadas en ambas figuras se ordenan siguiendo el mismo orden que las gráficas mostradas en la subsección 4.1.1: las columnas están ordenadas según la distribución de tamaño de tráfico (*Web Search* en la columna de la izquierda y *Data Mining* en la columna de la derecha) y las filas están ordenadas según el tipo de tráfico, es decir, los flujos del tipo *Elefante* se sitúan en la primera fila, los flujos del tipo *Conejo* en la segunda fila y los flujos del tipo *Ratón* en la tercera fila.

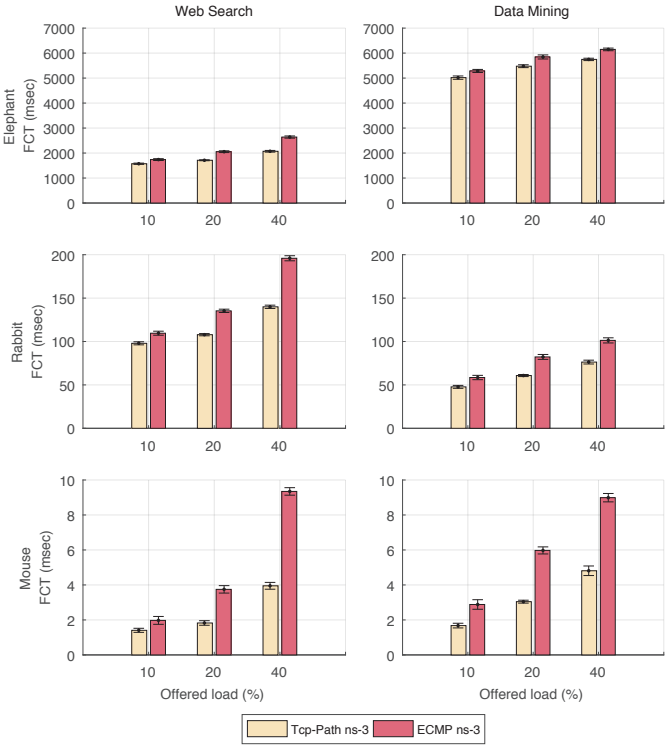
En dicha imagen (figura 5.7) se observa como el protocolo propuesto (TCP-Path) introduce una mejora de rendimiento respecto al algoritmo de referencia (ECMP), elevando el BW medio por flujo y, a la vez, disminuyendo el FCT medio de cada uno de ellos. Esta mejora, se debe a que TCP-Path establece las rutas en función del estado de la red en el momento de creación de cada uno de los flujos, es decir, el protocolo determina bajo demanda el camino cuya latencia es menor para cada flujo, mientras que ECMP establece los caminos fijando como parámetro de selección el número de saltos que tiene cada uno de ellos, sin considerar ninguna otra información adicional sobre el estado de la red.

Con el objetivo de tener una imagen más clara de las mejoras obtenidas tras comparar TCP-Path con ECMP se presenta la figura 5.8 [12]. En ella se observa como al aplicar el protocolo TCP-Path como mecanismo de encaminamiento sobre un tráfico cuya distribución de tamaño de tráfico es *Web Search*, se obtiene un aumento del BW por flujo que varía desde el 5 %, para los flujos de tipo *Ratón* al aplicar una carga del 10 %, y el 25 %, para los flujos de tipo *Elefante* aplicando una carga del 40 %. Del mismo modo, se observa una disminución del FCT medio que varía desde el 10 % para los flujos de tipo *Ratón* cuando la carga está fijada al 10 % y el 58 % para los flujos de tipo *Elefante* cuando la carga se encuentra al 40 %. Esta gráfica también muestra como, si se aplica el mismo protocolo (TCP-Path) sobre una distribución de tráfico cuya distribución de tamaño de flujo sigue la distribución *Data Mininig*, la mejora obtenida en el BW oscila entre el 9 % para los flujos de tipo *Ratón* cuando el sistema se carga al 10 % y el 18 % para los flujos del tipo *Elefante* con una carga ofrecida del 40 %. Del mismo modo, la figura muestra como el FCT medio decrece entre el 5 % para el caso de los flujos de tipo *Elefante* y el 50 % para los flujos de tipo *Conejo* a una carga del 20 %.

Para validar el modelo creado en el simulador ns-3, se decidió repetir las pruebas anteriores, pero utilizando en este caso un entorno de pruebas emulado. Para ello, se utilizó el emulador de red Mininet [82], cuya pila de *protocolos TCP* es la implementada en los sistemas reales con sistema operativo *Linux*. Estas se realizaron a una velocidad de transmisión de 10 Mbps puesto que Mininet al ser un emulador necesita de mayores



(a) Comparación en términos de BW Medio



(b) Comparación en términos de FCT

Figura 5.7: Comparación de rendimiento entre TCP-Path y ECMP [12]

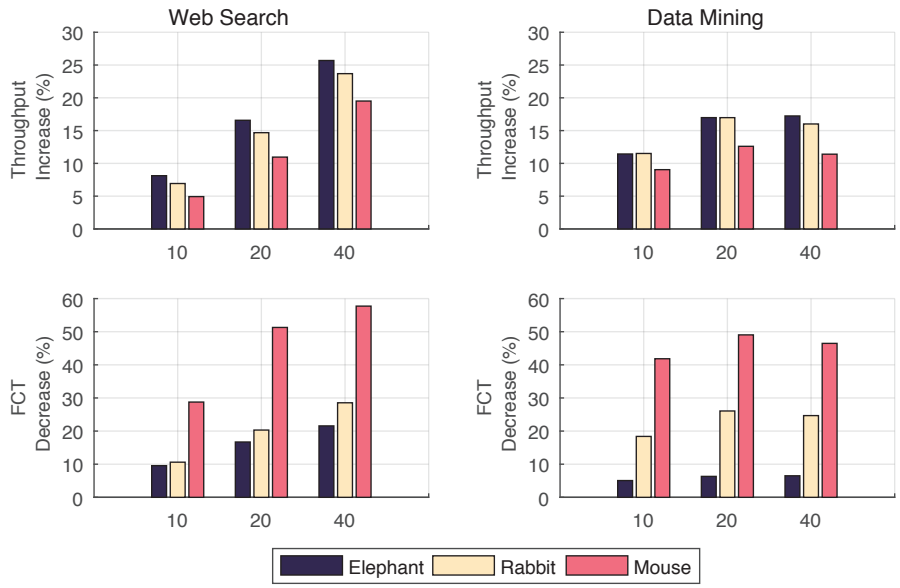
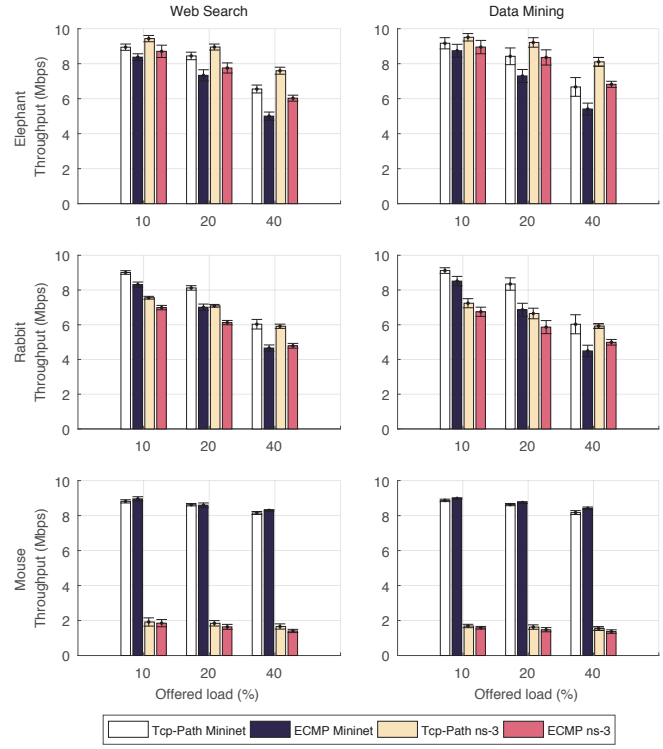


Figura 5.8: Mejora del rendimiento obtenido por TCP-Path respecto a ECMP representada en %

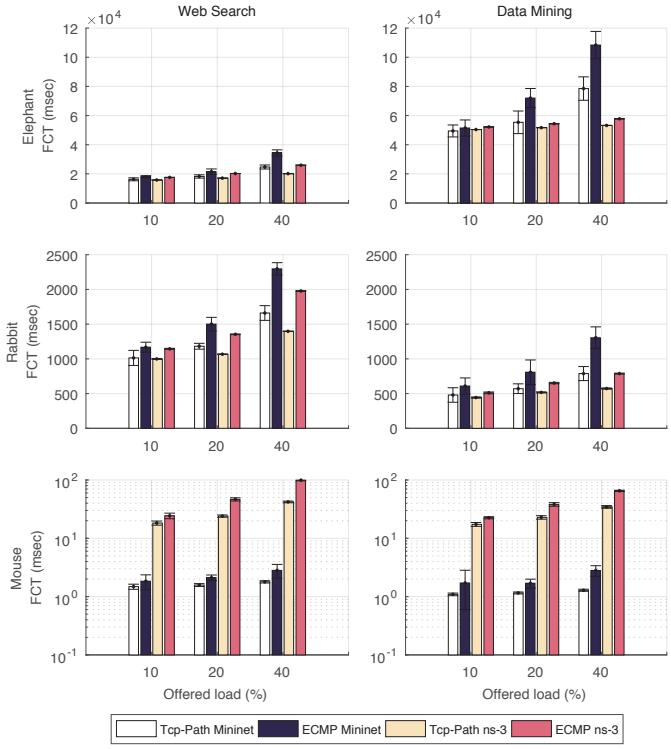
recursos para poder emular redes a 100 Mbps. Además, el objetivo es validar la implementación del simulador para poder dar como válidos los resultados explicados anteriormente.

Esta validación produjo los resultados representados en la figura 5.9 [12]. En esta imagen queda reflejado cómo los resultados obtenidos por el modelo simulado y el modelo emulado son muy cercanos para los flujos de tamaño mediano (flujos de tipo *Conejo*) y grande (flujos de tipo *Elefante*). Por el contrario, distan mucho en el caso de los flujos de tipo *Ratón*. Esta diferencia se produce porque la generación de las tramas no es la misma en ambos sistemas de pruebas, ya que en el entorno emulación la información que lleva un flujo se genera completamente en la capa de aplicación y es el *socket* el encargado de segmentarla. En cambio, debido a una limitación en la memoria RAM del entorno hardware donde se ejecuta el simulador, en éste, la información se genera trama a trama con una tasa constante de generación igual a la velocidad del enlace. Además, los algoritmos de control de congestión utilizados en el simulador y en el emulador no eran los mismos, para el caso del sistema emulado se utilizó *Cubic*, mientras que para el simulador el único disponible era *Vegas*. Estas diferencias en flujos grandes quedan muy diluidas, ya que el número de tramas que los componen es grande, pero queda muy patente en los flujos de tipo *Ratón*.

Al igual que en el caso de los resultados en entorno simulado, la figura 5.10 [12] muestra un resumen de la mejora obtenida (en valor porcentual), tanto para el caso del entorno simulación, como para el entorno emulado. En ella se observa como los resultados obtenidos para la distribución de tráfico que sigue el modelo *Web Search* son muy próximos tanto en la mejora de ancho de banda como en la reducción del FCT medio utilizado por



(a) Comparación en términos del BW medio



(b) Comparación en términos de FCT

Figura 5.9: Validación del modelo emulado

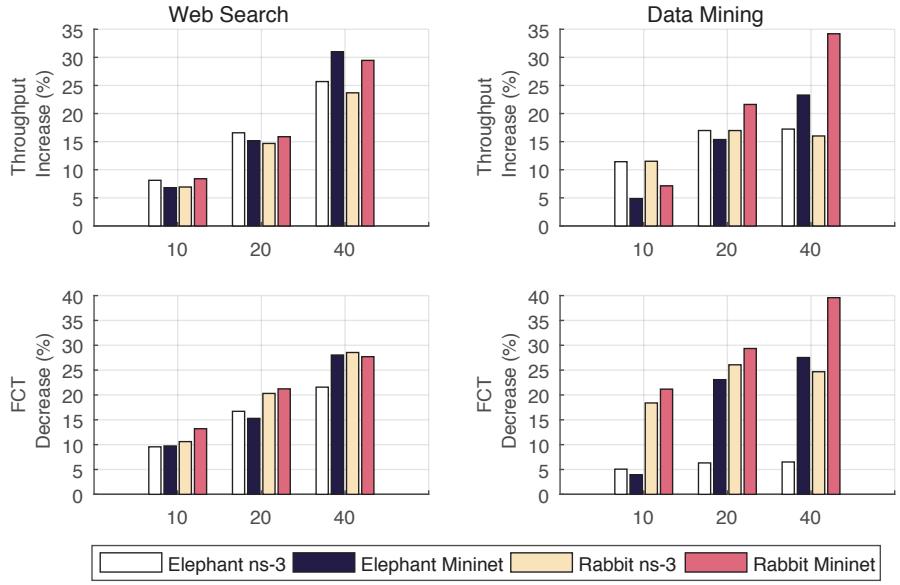


Figura 5.10: Mejora porcentual del BW medio obtenido por TCP-Path respecto a ECMP en los entornos de simulación y emulación a 10 Mbps

flujo. Por el contrario, en el caso de los resultados obtenidos para un tráfico que obedece a la distribución *Data Mining* se observa como existe una mayor diferencia entre ambos entornos, esto se debe que esta distribución basa casi todo su tráfico en flujos del tipo *Ratón*.

5.1.5.2 Tamaño de Tablas

TCP-Path basa su funcionamiento en el establecimiento de un nuevo camino para cada flujo TCP que aparece en la red, es decir, crea un nuevo camino para aproximadamente el 95 % de los flujos que recorren las redes actualmente. Para llevar a cabo esto, TCP-Path realiza un proceso de exploración y confirmación cada vez que un flujo TCP aparece en la red lo que provoca que se genere un nuevo registro en la tabla de encaminamiento del switch por cada nuevo flujo TCP. Por lo tanto, se puede concluir que el número de entradas en las tablas de encaminamiento aumentará cuando el número de flujos TCP aumente, lo que puede provocar que el protocolo no escale de forma correcta.

Para comprobar esta premisa se han realizado pruebas donde se mide el número de entradas media que TCP-Path tiene en su tabla de encaminamiento. Para ello se ha utilizado el mismo juego de pruebas que para las pruebas de rendimiento (ver tabla 5.2 a modo de resumen). Para la realización de esta prueba se ha fijado un tiempo de vida medio de los registros en tabla de 10 segundos, este tiempo es el máximo tiempo que una entrada podrá estar en la tabla sin actualizarse, es decir, sin que se haya utilizado para el encaminamiento de algún flujo.

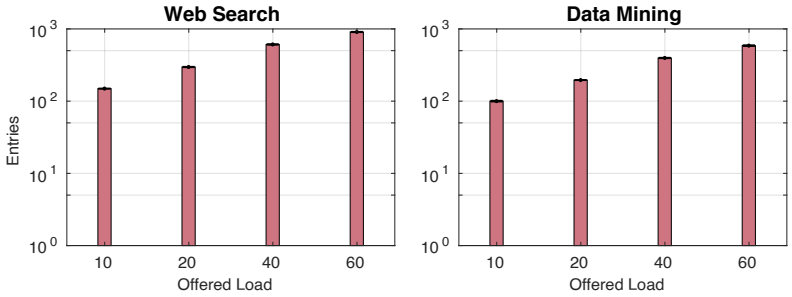


Figura 5.11: Tamaño de tabla de encaminamiento de TCP-Path

En la figura 5.11 [12] se muestra como el número de entradas en la tabla de encaminamiento aumenta según aumenta el número de flujos TCP que recorren la red, es decir, a mayor número de flujos mayor será el número de entradas. Este efecto puede provocar varias situaciones comprometidas: la primera, aumentar el tiempo necesario para procesar la entrada-salida de las tramas provocando una disminución del rendimiento de la red, segundo, que sea necesario disponer de una cantidad de memoria excesiva ya que al tener que almacenar tanto estado si la memoria es pequeña puede producir que no se guarden todas las reglas necesarias y, por lo tanto, el encaminamiento de los flujos no sea correcto. En resumen, si el número de entradas en la tabla aumenta linealmente con el número de flujos que recorren la red (una entrada por cada flujo TCP nuevo) puede producirse un problema de escalabidad en los sistemas.

5.1.5.3 Difusión de tramas

Como se ha explicado, el funcionamiento del protocolo se basa en la difusión de tramas para la realización de las distintas exploraciones. Estas exploraciones pueden ser un problema cuando el sistema esté cargado, por lo que se ha decidido realizar una evaluación para comprobar la escalabilidad del protocolo debida a la difusión de tramas necesarias para dichas exploraciones [14]. Para analizar este tráfico extra (tramas *Path Request*), se va a considerar una topología *Spine-Leaf* compuesta por switches *Spine*, a partir de ahora N_S, y switches *Leaf*, a partir de ahora N_L.

El proceso de difusión para generar una ruta de encaminamiento para un flujo *TCP*, según TCP-Path, comienza cuando un N_L encapsula el mensaje *TCP-SYN* en una trama de control *Path Request* con el objetivo de difundirla por todos sus puertos (excepto el de entrada). Estas tramas serán recibidas por los N_S, los cuales las difundirán, a su vez, por todos sus puertos menos por el de recepción. El proceso de difusión termina cuando las tramas *Path Request* llegan a los N_L, los cuales desencapsulan la trama obteniendo el mensaje *TCP-SYN* original. Por lo tanto, se puede definir el número total de *bytes* de control difundidos que un N_L procesa en media como el número total de flujos TCP que recorren la red en *bytes* (número de flujos N_F por tamaño en *bytes* de cada flujo L_B [64 *bytes*]), multiplicado por el producto entre el número total de N_S, el número total de N_L menos uno y dividido por el número de N_L que existen en la red (ver ecuación 5.1):

$$Bcast_{por_{LEAF}} = \frac{N_S \cdot (N_L - 1)}{N_L} \cdot L_B \cdot N_F \tag{5.1}$$

Así mismo se puede calcular el tráfico de tramas *unicast* que procesa cada N_L como el doble del número de flujos TCP que recorren la red por el tamaño medio de flujo en *bytes* $\overline{L_F}$ y todo ello divido por N_L (ver ecuación 5.2).

$$Ucast_{por_{LEAF}} = \frac{2 \cdot N_F}{N_L} \cdot \overline{L}_F \tag{5.2}$$

En definitiva, el ratio de *bytes* de difusión utilizados por el protocolo respecto a la cantidad de *bytes* de tramas unicast sería (ver ecuación 5.3) la división entre el número de *bytes* de difusión por *Leaf* (ecuación 5.1) y el número de *bytes* unicast por *Leaf* (ecuación 5.2).

$$\frac{Bcast_{por_{LEAF}}}{Ucast_{por_{LEAF}}} = \frac{N_S \cdot (N_L - 1)}{2} \cdot \frac{L_B}{\overline{L}_F} \tag{5.3}$$

Tomando 64 *bytes* como tamaño de la trama de difusión, 1.90 *Mbytes* como tamaño medio de para la distribución *Web Search*, 2.97 *Mbytes* para la distribución *Data Mining* y variando el número total de N_S o N_L en la ecuación 5.3, se obtienen los resultados mostrados en la tabla 5.3. Como muestran los resultados, el porcentaje de *bytes* utilizados por el mecanismo de exploración aumenta con el número de switches que componen la red, pasando de un tráfico despreciable en redes pequeñas (4 *Leaf* y 4 *Spine*) a un tráfico muy considerable (27.49 %) en redes de un tamaño mayor (64 *Leaf*, 256 *Spine*).

Como validación de los cálculos teóricos se ha realizado una simulación con enlaces a 1 Gbps en una topología *Spine-Leaf* con 4 *Leaf*, 4 *Spine* y 20 *host* por *Leaf* con diferentes cargas (10 %, 20 %, 40 % y 60 %) en las dos distribuciones de tráfico (*Web Search* y *Data Mining*). Los resultados obtenidos y, mostrados en la figura 5.12, corroboran los resultados teóricos ya que muestran que una red poco cargada y formada por 4 *Leaf* y 4 *Spine* tiene un porcentaje de difusión de aproximadamente 0.020 %, pero también muestran que, para un mismo tamaño de red, el tráfico de difusión utilizado aumenta a medida que aumenta la carga ofrecida.

Distribución	N_S	N_L	**TCP-Path**
Web Search	4	4	0.02
	16	64	1.70
	32	128	6.84
	64	256	27.49
Data Mining	4	4	0.01
	16	64	1.09
	32	128	4.38
	64	256	17.58

Tabla 5.3: Calculo teórico del ratio de *bytes* de difusión utilizado por TCP-Path respecto al número de *bytes* de tramas unicast (%)

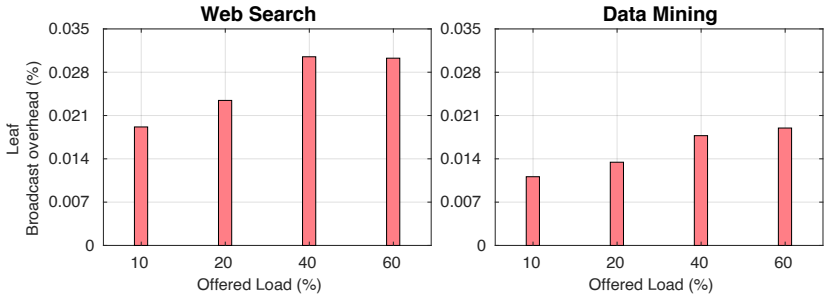


Figura 5.12: Porcentaje de difusión medio por Leaf

5.2 TFE: TCP-Path para *Elefantes*

Para resolver los posibles problemas de escalabilidad mencionadas en la sección 5.1, se ha diseñado TFE, una variante cuyo objetivo es paliar dichos problemas. Para ello, TFE aplica únicamente el mecanismo de descubrimiento de caminos a un subconjunto de los flujos.

Analizando las dos distribuciones de tamaño de tráfico, *Web Search* [83] y *Data Mining* [84], utilizadas en las pruebas (figura 4.3), se observa como los flujos de menor tamaño, flujos de tipo *Ratón* y flujos de tipo *Conejo*, ocupan más del 85 % de las conexiones TCP (figura 4.4), mientras que respecto al tamaño en *bytes* apenas representan 5 % del total. Por otro lado, también se observa como los flujos de mayor tamaño, flujos de tipo *Elefante*, aportan más del 90 % de los *bytes* que recorren la redes, mientras que en porcentaje de conexiones no superan el 5 % del total. En conclusión, no es necesario tratar todos los flujos TCP sino que solo es necesario tratar de manera especial los flujos de gran tamaño, ya que, aunque pocos en número, pueden llegar a saturar o mermar el rendimiento de la red.

Es por este motivo, por el que nace TFE. Esta variante de TCP-Path tiene como objetivos balancear de la mejor manera posible los flujos de gran tamaño o flujos de tipo *Elefante*, y disminuir el tráfico de difusión generado por TCP-Path.

5.2.1 Diseño funcional de TFE

TFE establece algunos cambios en la lógica funcional de TCP-Path. El primero y más importante, es que el switch frontera que recibe la primera trama del flujo TCP, es decir, el *TCP-SYN*, no solo debe comprobar que dicho flujo es un flujo TCP, sino también debe comprobar que dicho flujo es de tipo *Elefante*. Como se representa en la figura 5.13.

En el caso de cumplir ambas condiciones, se debe aplicar la misma lógica definida por el protocolo TCP-Path y explicada en la sección 5.1.1. Por el contrario, si no cumple alguna de las dos condiciones, el flujo será encaminado por el protocolo de encaminamiento que el switch tenga implementado por defecto. En el caso de esta tesis dicho protocolo será ARP-Path [44].

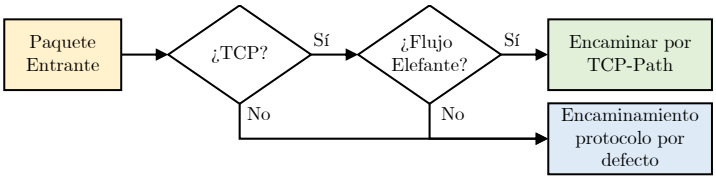


Figura 5.13: Diagrama de flujo TFE

5.2.2 Implementación de TFE en un switch SDN

El protocolo TFE establece una evolución del protocolo TCP-Path con el objetivo de solventar los problemas de escalabilidad de TCP-Path, pero sin perder rendimiento. Por ello, TFE propone emplear la exploración de caminos exclusivamente para un subconjunto de los flujos, que debe ser preconfigurado por el administrador de la red. Para el caso de estudio realizado en esta tesis, se ha establecido que únicamente TFE entre en juego para los flujos de tipo *Elefante*. Los flujos de tipo *Elefante* siempre se enviarán en nuestros entornos de pruebas usando un puerto TCP destino superior al *"30000"* mientras el resto de los flujos que usan algún puerto TCP por debajo de dicho valor.

Al tratarse de una evolución, la implementación necesaria para llevar a cabo TFE se puede ver como una variación de la implementación de TCP-Path (Ver apartado 5.1.3). Esta modificación se muestra en el algoritmo 5.10. Como se puede observar en la figura 5.13, el sistema necesita identificar si una trama debe o no ser tratada por el nuevo protocolo ya sea porque es una trama propia del protocolo, o porque se trata de una trama que pertenece a uno de los flujos elegidos por el administrador. Para comenzar,

```
Data: Trama entrante; Puerto Umbral
Result: Aplicación de TCP-Path o mecanismo de encaminamiento por defecto
begin
    if Trama->IP->Protocolo == '6' and Trama->TCP != NULL then
        if Trama->TCP->Port->Dst >= Puerto Umbral or Trama->TCP->Port->Scr >=
          Puerto Umbral then
            if Trama->TCP->Flags == 'SYN+ACK' then
                Encapsular en 'Path Reply';
            else if Trama->TCP->Flags == 'SYN' then
                Encapsular en 'Path Request';
            else
                Encaminar 'Trama Unicast';
        else
            Encaminar por protocolo de encaminamiento por defecto;
    if  Trama->TCP-PATH != NULL then
        if Trama->TCP-PATH->Tipo == 'Request' then
            Tratar la trama 'Path Request';
        else if Trama->TCP-PATH->Tipo == 'Reply' then
            Tratar la trama 'Path Reply';
        else
            Descartar trama errónea;
    else
        Encaminar por protocolo de encaminamiento por defecto;
```

Algoritmo 5.10: Procesamiento general de las tramas en TFE

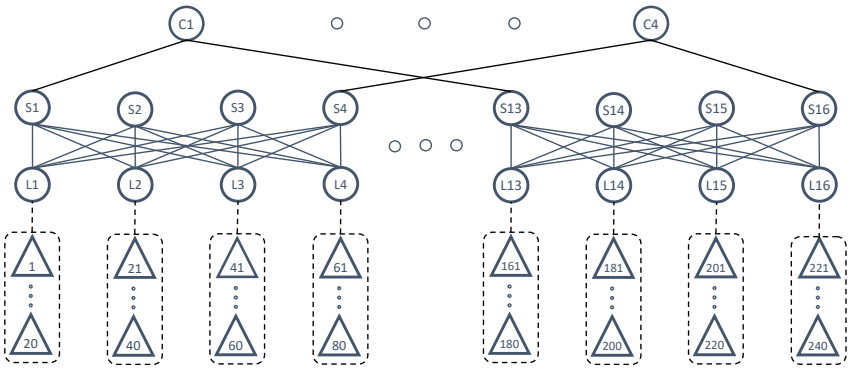


Figura 5.14: Topología tres niveles basada en Altoona

el switch comprueba si la trama tiene como protocolo de transporte TCP y, también, si su cabecera TCP es válida. En caso afirmativo, comprueba si la trama pertenece a uno de los subconjuntos de flujos seleccionados, para ello comprueba el valor de su puerto destino y de su puerto origen. La comprobación del puerto destino se debe a que, como se han explicado en párrafos anteriores, se ha fijado que todos los flujos que superen el puerto *'30000'* deben ser tratados por TFE; mientras que la comprobación del puerto origen se debe a que el protocolo TCP establece que las tramas de datos deben ser confirmadas, por lo tanto, existe un flujo de tramas en sentido contrario, lo que provoca que estas tramas también deban ser tratados por TFE. En caso de que la comprobación sea correcta, la trama se trata como indica el protocolo TCP-Path (ver subsección 5.1.3.1). Por el contrario, si la trama recibida es propia del protocolo, TFE la trata como se ha explicado en la subsección 5.1.3.3 del protocolo TCP-Path. En caso contrario, es decir, la trama no es ni una trama propia del protocolo, ni una trama del subconjunto de flujos seleccionados, se envía al protocolo de encaminamiento por defecto.

5.2.3 Evaluación

La evaluación de TFE ha sido realizada mediante una comparación en términos de BW y FCT entre TFE, TCP-Path y, como algoritmo de referencia, ECMP, ofreciendo a la red diferentes cargas de tráfico. También se ha comparado el tamaño de las tablas y el porcentaje de *bytes* de difusión entre TFE y TCP-Path.

Para la realización de las pruebas se han utilizado las topologías: *Spine-Leaf*, compuesta por 4 *Spine*, 4 *Leaf* y 20 equipos por *Leaf* (ver figura 4.2 [12]) y una topología de tres niveles basada en Altoona [90], compuesta por 4 *Cores*, 16 *Agregados*, 16 *ToRs* y 20 equipos por *ToR* (ver figura 5.14 [14]) junto a una matriz de tráfico aleatoria cuya única condición es que el equipo origen y el equipo destino de cada flujo deben estar alojados en diferentes *Leafs*/*ToRs*. Se han utilizado, también, al igual que en las pruebas presentadas para TCP-Path, dos tipos diferentes de distribuciones de tamaño de flujos, *Web Search* y *Data Mining* (ver figura 4.3 [12]). Como resumen de los parámetros de configuraciones utilizados en las pruebas se presenta la tabla 5.4.

Característica	Valor
Topología de la red	Spine-Leaf (4 - 4) [85] y Topología tres capas (4-16-16) (Derivada de la topología Altoona de Facebook)
Equipos por switch de nivel 1	20
Matriz de Tráfico	Aleatoria entre switches de nivel 1
Distribuciones tamaño de flujos	Web Search[83] & Data Mining [84]
Carga ofrecida a la Red (%)	10, 20, 40 & 60 %
Velocidad de los enlaces	10 Mbps (emulación y simulación), 100Mbps & 1Gbps (Simulación)
Tiempo de prueba (s)	1800 s
Tiempo Transitorio (s)	800 s
Número de repeticiones	10

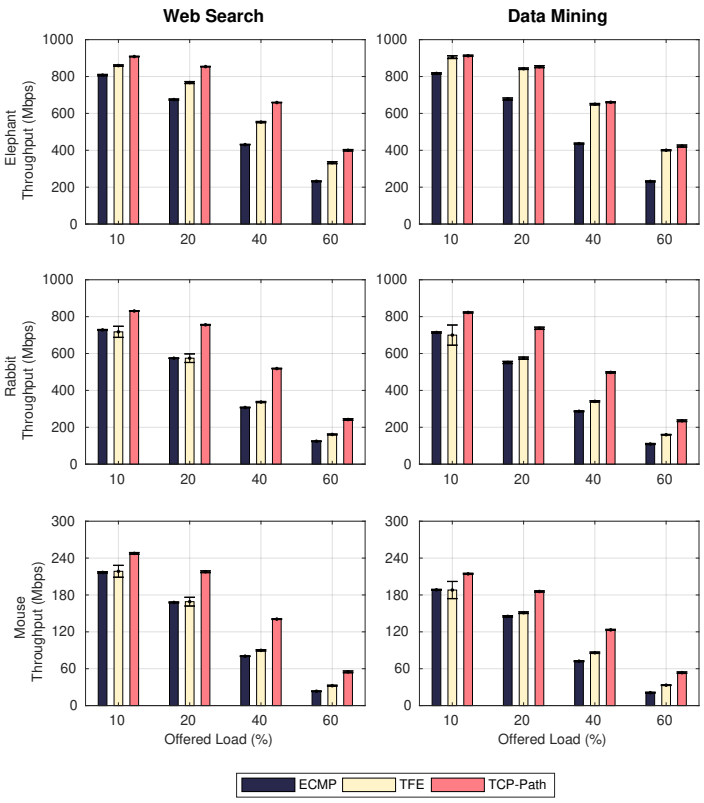
Tabla 5.4: Configuración para evaluación

Por último, para la realización de las pruebas se ha contado con la misma infraestructura hardware que en las pruebas realizadas al protocolo TCP-Path (ver sección 5.1.5.1). Se ha utilizado como software de simulación ns-3 en su versión 26 con BIC como mecanismo de control de congestión y como software de emulación Mininet [82] con *Cubic* como control de congestión. Al igual que en las pruebas anteriores (ver sección 5.1.4) el uso de dos plataformas distintas para el mismo banco de pruebas tiene como finalidad la validación del modelo simulado (ns-3) por un modelo emulado (Mininet).

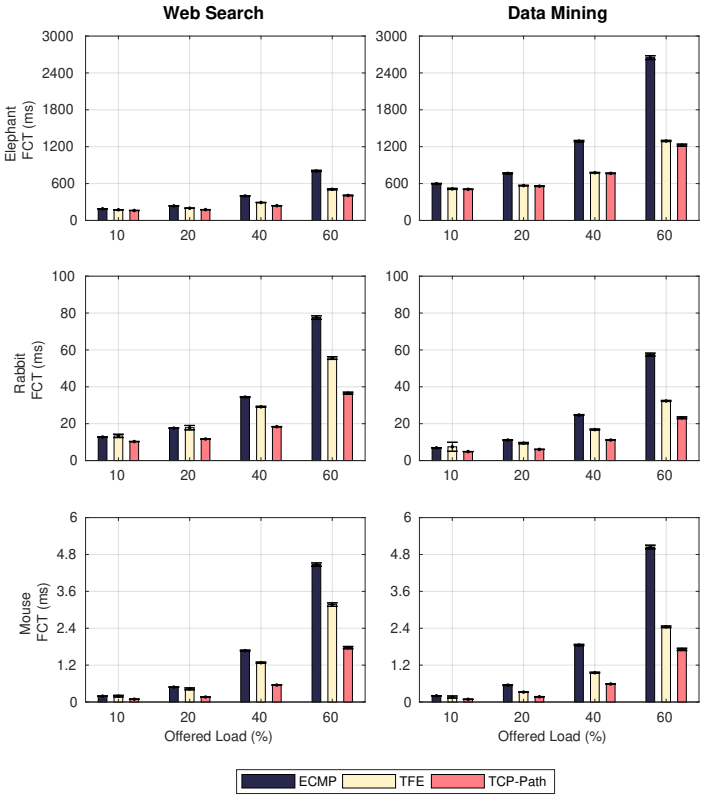
5.2.4 Rendimiento respecto a ancho de banda medio (BW) y Flow Completion Time (FCT)

Los resultados obtenidos para las pruebas realizadas en la topología *Spine-Leaf (4x4)* con velocidad de transmisión de 1 Gbps en cada enlace, se han representado en la figura 5.15 [14], mientras que los resultados obtenidos para las pruebas realizadas en la topología de tres niveles basada en la topología *Altoona* con 4 *Cores*, 16 *Agregados* y 16 *ToRs*, con enlaces cuya velocidad de transmisión se ha fijado a 10 *Mbps*, se muestran en la figura 5.17 [14] siguiendo el mismo esquema que el presentado para TCP-Path. En la topología *Spine-Leaf* se observa (ver figura 5.15 [14]) cómo el rendimiento de TCP-Path (tanto en BW por flujo como en FCT) es el más alto de los tres protocolos comparados, pero también se observa como TFE, aun teniendo un rendimiento inferior a TCP-Path, supera en rendimiento al algoritmo estándar para encaminamiento (ECMP).

Para tener una imagen más clara de la mejora obtenida por los protocolos TCP-Path y TFE respecto a ECMP se ha incluido la figura 5.16 [14]. En ella se muestra como en flujos de gran tamaño (flujos de tipo *Elefante*) la mejora porcentual de TCP-Path y TFE es muy similar para ambas distribuciones de tráfico. Esto se debe a que ambos protocolos se comportan de la misma manera con estos flujos, se recuerda que TFE intenta solventar los problemas de escalabilidad mostrados por TCP-Path sin perder el buen rendimiento mostrado por él. Sin embargo, en los flujos de tamaño medio (flujos de tipo *Conejo*) y tamaño pequeño (flujos de tipo *Ratón*) la mejora es mucho mayor cuando se aplica TCP-Path como protocolo de encaminamiento. Esto se debe a que en esta ocasión TFE no se comporta como dicho protocolo sino que al delegar el encaminamiento de este tipo

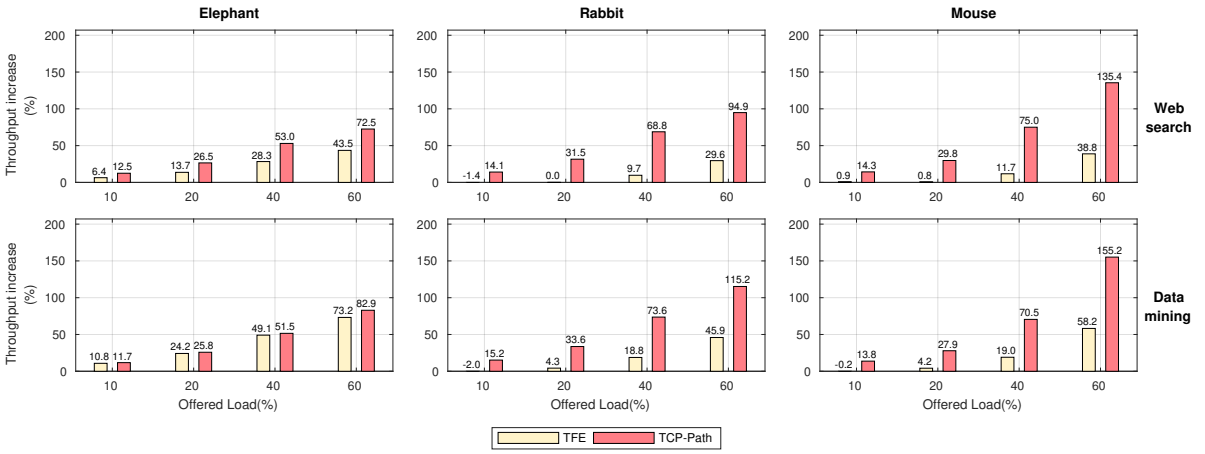


(a) Comparación en términos del BW

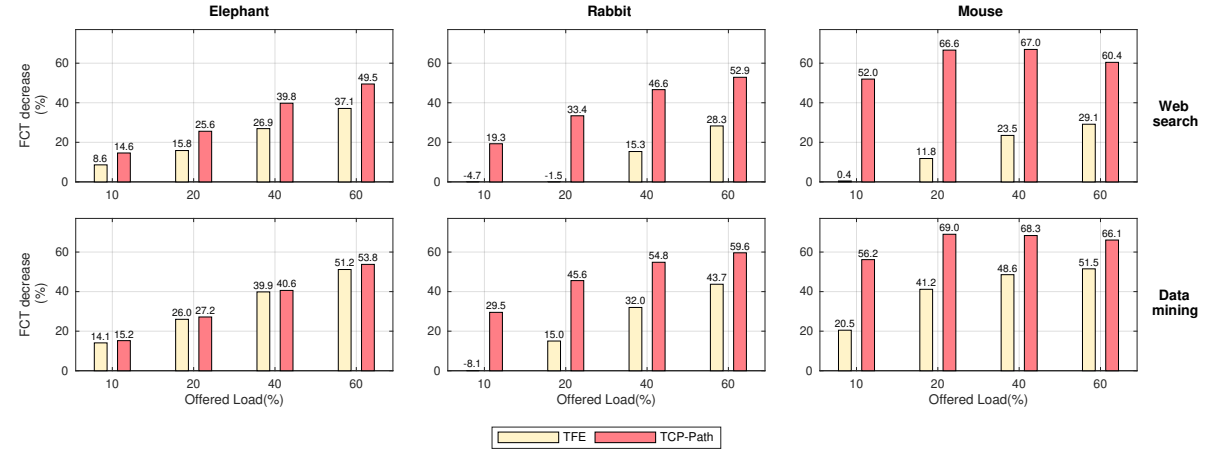


(b) Comparación en términos de FCT

Figura 5.15: Comparativa de rendimiento en la topología Spine-Leaf 4x4 con enlaces a 1 Gbps [14]



(a) Comparación en términos del BW medio

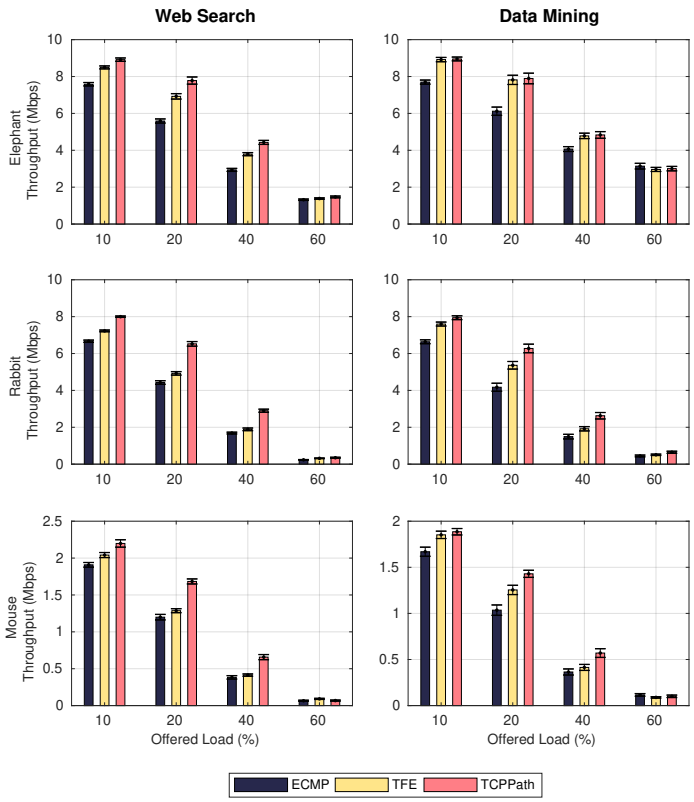


(b) Comparación en términos de FCT

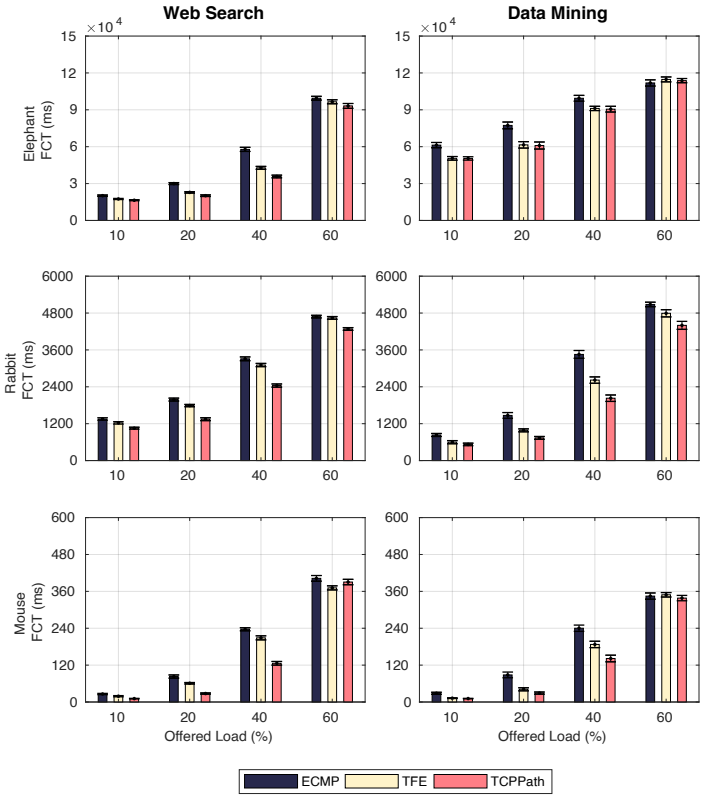
Figura 5.16: Porcentaje de Mejora de rendimiento obtenido en la topología Spine-Leaf 4x4 a 1 Gbps respecto a ECMP

de tráfico al protocolo de encaminamiento por defecto (ARP-Path) adquiere las ventajas y desventajas de este. La principal ventaja es la sencillez en el proceso de encaminamiento y su principal desventaja generar caminos que para altas cargas no producen un balanceo óptimo ya que quedan fijados de forma casi estática al comienzo de la simulación, cuando la red todavía se encuentra en un estado transitorio y con baja carga.

Por otro lado, también se puede observar como los resultados obtenidos para la topología *Spine-Leaf* se mantienen en la topología de tres niveles basada en la topología *Altoona* utilizada por *Facebook* (ver la figura 5.17 [14]). Esto se debe a que la topología de 3 niveles sigue manteniendo el mismo número de caminos posibles que en la topología *Spine-Leaf*, es decir, cada switch tiene la posibilidad de elegir entre cuatro posibles enlaces con la capa superior y tres posibles enlaces con la capa inferior para establecer la nueva ruta. Si, por el contrario, se estableciera como topología la *VL2* [84] donde un switch *ToR* solo dispone de dos caminos disjuntos entre un origen y un destino, el rendimiento de los protocolos propuestos (TCP-Path y TFE) se aproximaría al rendimiento obtenido



(a) Comparación en términos del BW medio

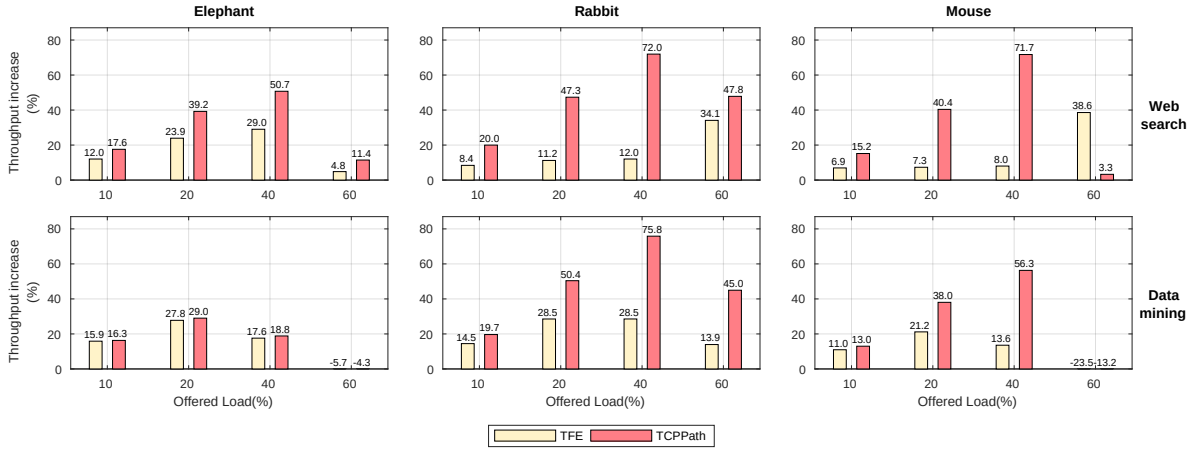


(b) Comparación en términos de FCT

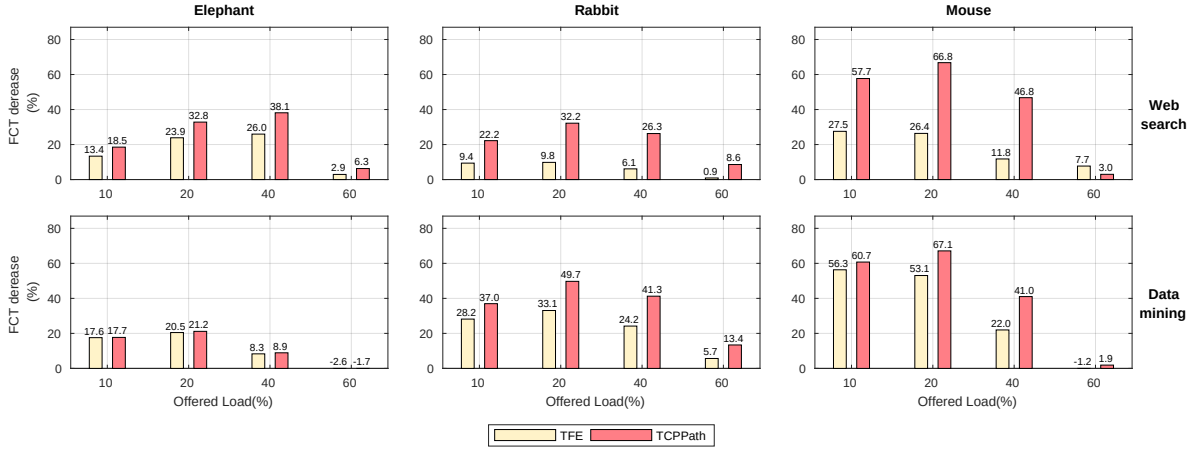
Figura 5.17: Comparativa de rendimiento en la topología de tres niveles a 10 Mbps [14]

por el algoritmo de referencia (*ECMP*). Este descenso del rendimiento viene dado por la diferencia de caminos disjuntos que existe entre la topología *VL2* y las topologías tipo utilizadas, es decir, en el caso de la *VL2* el número de caminos disjuntos es únicamente de dos mientras que en el caso de la *Spine-Leaf* es mucho mayor (tantos como nodos *Spine*). Este número tan bajo de caminos disjuntos en la topología *VL2* tiene como consecuencia que la probabilidad de que un camino se elija varias veces sea del 50 % y, por lo tanto, la posibilidad de que dos flujos de tipo *Elefante* confluyan en la misma ruta es alta, dando lugar a posibles bajadas de rendimiento debido a un mal balanceo de la carga.

La mejora de rendimiento obtenida por el protocolo TFE respecto al protocolo de referencia (ECMP) se observa con más claridad en la figura 5.18 [14], donde se ha representado el ratio de mejora porcentual de los protocolos TFE y TCP-Path respecto al rendimiento ofrecido por el algoritmo de referencia ECMP. En estas imágenes se muestra que al cuanto mayor sea la carga ofrecida al sistema, mejor se comportan TCP-Path y TFE respecto a ECMP. Este aumento del rendimiento por parte de TFE llega a alcanzar



(a) Comparación en términos del BW medio



(b) Comparación en términos de FCT

Figura 5.18: Porcentaje de Mejora de rendimiento obtenido en la topología de tres niveles a 10 Mbps respecto a ECMP [14]

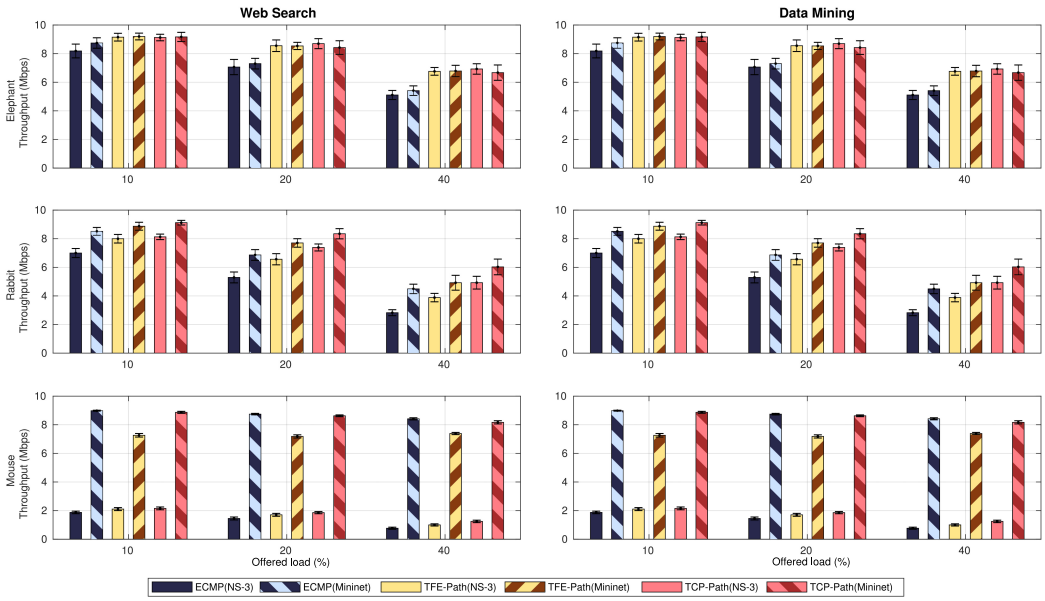


Figura 5.19: Comparativa de BW en la topología Spine-Leaf 4x4 entre simulador y emulador a 10 Mbps

un 73.2 % de mejora del ancho de banda medio al 60 % de carga en flujos del tipo *Elefante* para una distribución *Data Mining*, del mismo modo en el caso de la distribución *Web Search* se alcanza una mejora del 43.4 % para el mismo tipo de flujos. Asimismo, para el caso del FCT se observa que en una distribución *Data Mining* se obtiene una mejora máxima del 51.5 % en flujos del tipo *ratón* con un 60 %, mientras que para la distribución *Web Search* el máximo de mejora se obtiene para flujos de tipo *Elefante* también con un 60 % de carga ofrecida.

Siguiendo la misma dinámica que en el apartado 5.1.4, se han realizado una serie de pruebas para validar el modelo simulado realizando para ello una comparación entre los resultados obtenidos en dicho modelo con los obtenidos en un modelo emulado donde las implementaciones de las pilas TCP son reales. Por esta razón, se ha repetido el juego de pruebas con la topología *Spine-Leaf(4x4)* en Mininet. En este conjunto de pruebas los enlaces que unen los diferentes switches tienen fijada una velocidad de transmisión de 10 Mbps en ambos entornos, el uso de una velocidad baja se debe a la restricción que existe en el hardware disponible en el entorno de emulación.

En estas pruebas se obtuvieron los resultados que han sido representados en la figura 5.19 [14]. En dicha figura se observa como en el caso de flujos de gran tamaño (flujos de tipo *Elefante*) y flujos de tamaño medio (flujos de tipo *conejo*) los resultados en ambos entornos son muy aproximados. Por el contrario, para los flujos de pequeño tamaño (flujos de tipo *Ratón*) se aprecia una gran diferencia de rendimiento. Esta diferencia se debe a la manera que tiene cada uno de los entornos de pruebas para crear las tramas de un flujo, y ya ha sido ha sido explicado en la subsección 5.1.5.1.

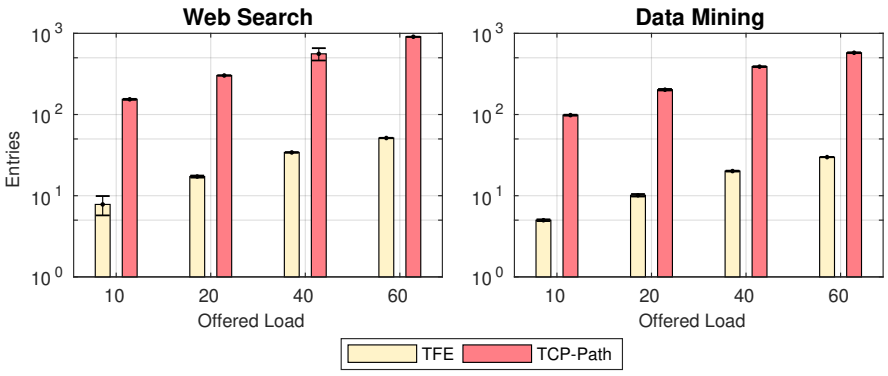


Figura 5.20: Tamaño de las tablas de los switches en la topología Spine-Leaf 4x4 a 1 Gbps

5.2.5 Tamaño de las Tablas

TFE, a diferencia de TCP-Path (ver apartado 5.1.5.2), solo genera una exploración por cada flujo de tipo *Elefante* (aproximadamente el 5 % de los flujos considerando las distribuciones de tamaño de tráfico utilizadas para las pruebas (*Web Search* y *Data Mining*), ya que el resto de flujos son encaminados por el protocolo por defecto, ARP-Path. Esto provoca que el número de entradas en la tabla sea mucho menor, ya que solo se tiene una entrada por cada flujo del tipo *Elefante*, además de una entrada por cada equipo final de la red. Para validar este razonamiento se ha realizado una medida del número de entradas media que TCP-Path y TFE tienen en su tabla de encaminamiento mediante el uso del mismo juego de pruebas utilizado para las pruebas de rendimiento. Se establece el mismo tiempo de vida de una entrada que el establecido en el apartado 5.1.5.2, es decir, 10 segundos.

En la figura 5.20 [14] se muestra cómo el número de entradas en la tabla de encaminamiento para TFE es mucho menor que el número de entradas para TCP-Path. Esta disminución del número de entradas en la tabla se debe a que TFE solo genera un camino nuevo para cada flujo de tipo *Elefante* (aproximadamente un 5 % del total de los flujos según las distribuciones de tamaño de tráfico utilizadas). Esta disminución facilita que el número de entradas en las tablas, aunque depende del número de flujos de tipo *Elefante*, sea muy manejable para los dispositivos, por lo que el tiempo de búsqueda en la tabla será mucho menor que en TCP-Path. De esta manera, la escalabilidad del sistema no se vea comprometida.

5.2.6 Sobrecarga por difusión de tramas de control

Uno de los objetivos principales de TFE es reducir la cantidad de *bytes* difundidos por el protocolo TCP-Path. Es por ello que, siguiendo con la idea presentada en la sección 5.1.5.3, se va a estudiar la cantidad de *bytes* de difusión utilizado por cada uno de los protocolos en diferentes topologías jerárquicas.

Extendiendo el estudio teórico realizado en la sección 5.1.5.3 a topologías de 3 capas y fijando dicho estudio para las topologías *VL2* [84] y *PortLand* [91], se puede obtener que para el caso de la topología *VL2*, es sabido que el número de *ToRs* que la componen es directamente proporcional al número de puertos que tienen sus *Cores* (K_C) y al número de puertos que tienen sus *Agregados* (K_A) y, que por su estructura, los ToR únicamente están conectados a dos de los switches *Agregados*, por lo tanto, el número de tramas de difusión que va a recibir cada uno de los ToR es de dos por cada flujo. Por tanto, la expresión matemática que cuantifica la cantidad de difusión (en *bytes*) en una topología *VL2* es la ecuación 5.4.

Mientras que, en el caso de la topología *PortLand*, es sabido que el número de puertos de todos los switches de la red es el mismo (K) y que el número de switches ToR depende de forma cuadrática del número de puertos que tengan los equipos y del número de *PoDs*, siendo un *PoD* la asociación de un conjunto de switches *agregados* y switches ToR. Para calcular la expresión matemática que cuantifica la cantidad de difusión en esta topología se ha de separar en dos partes, por un lado, el cálculo del tráfico de difusión del primero *PoD* y, por otro lado, el cálculo para resto de *PoDs*. El tráfico del primer *PoD* se puede calcular como el producto del número de *Agregados* que lo forman por el número de ToR que tiene menos uno, mientras que para el resto de *PoDs*, es el producto del número de switches *Cores* por el número de ToRs del *PoD*. La expresión general es la suma de ambos cálculos (ver 5.5).

$$Bcast_{ToR_{VL2}} = 2 \cdot (\frac{(K_A \cdot K_C) - 4}{(K_A \cdot K_C)}) \cdot N_F \cdot L_F \tag{5.4}$$

$$Bcast_{ToR_{PortLand}} = \frac{K^2 + 2 \cdot K + 4}{4 \cdot K} \cdot N_F \cdot L_F, \tag{5.5}$$

Sustituyendo el número de switches ToRs por la expresión matemática que lo relaciona con el número de puertos de los dispositivos en la expresión 5.2, la cual nos permite calcular cantidad de tráfico (en *bytes*) que trata un switch ToRs. Se pueden obtener las expresiones 5.6 y 5.7 para representar el ratio de tráfico de difusión (en *bytes*) respecto al tráfico unicast (en *bytes* que generan los protocolos propuestos (TCP-Path y TFE).

$$\frac{Bcast_{ToR_{VL2}}}{Ucast_{TOR_{VL2}}} = \frac{(K_A \cdot K_C) - 4}{4} \cdot \frac{L_B}{\overline{L_F}} \tag{5.6}$$

$$\frac{Bcast_{ToR_{PortLand}}}{Ucast_{TOR_{PortLand}}} = \frac{K \cdot (K^2 + 2 \cdot K + 4)}{16} \cdot \frac{L_B}{\overline{L_F}} \tag{5.7}$$

TFE, a diferencia de TCP-Path, no genera un nuevo camino para cada flujo TCP, sino que genera un camino para cada flujo de tipo *Elefante*, es decir, solo genera un nuevo camino en aproximadamente el 5 % de los casos. Por lo tanto, los resultados teóricos

Topología	Distribución	N_S	N_L	TCP-Path	TFE
Spine-Lead	**Web Search**	4	4	0.02	0.001
		16	64	1.70	0.085
		32	128	6.84	0.342
		64	256	27.49	1.374
	Data Mining	4	4	0.01	0.001
		16	16	1.09	0.054
		64	64	4.38	0.219
		128	128	17.58	0.879

Tabla 5.5: Calculo teórico del ratio de *bytes* de difusión en relación con la cantidad de *bytes* de tráfico unicast utilizado por TFE en comparación con TCP-Path en topología de dos niveles (%)

Topología	Distribución	K_C	K_A	TCP-Path	TFE
VL2	**Web Search**	4	4	0.01	0.001
		16	64	0.86	0.043
		32	128	3.45	0.172
		64	256	13.79	0.690
	Data Mining	4	4	0.006	0.0003
		16	64	0.55	0.027
		32	128	2.20	0.110
		64	256	8.82	0.441

Tabla 5.6: Calculo teórico del ratio de *bytes* de difusión en relación con la cantidad de *bytes* de tráfico unicast utilizado por TFE en comparación con TCP-Path en topologías *Vl2* (%)

Topología	Distribución	K	TCP-Path	TFE
PortLand	**Web Search**	4	0.02	0.001
		16	0.98	0.049
		64	7.34	0.367
		128	56.94	2.847
	Data Mining	4	0.01	0.001
		16	0.62	0.031
		64	4.70	0.235
		128	36.43	1.821

Tabla 5.7: Calculo teórico del ratio de *bytes* de difusión en relación con la cantidad de *bytes* de tráfico unicast utilizado por TFE en comparación con TCP-Path en topologías PortLand (%)

obtenidos, a partir de las expresiones matemáticas 5.3, 5.6 y 5.7, en comparación con TCP-Path, son los representados en las tablas 5.5, 5.6 y 5.7.

Los resultados teóricos muestran un gran decremento del porcentaje de *bytes* de difusión utilizado por TFE. Para corroborar los buenos resultados obtenidos de forma teórica, se realizó una comprobación experimental, siguiendo el modelo de la sección 5.1.5.3, pero variando la capacidad de los enlaces, pasando de una velocidad inicial de 100 Mbps a 1 Gbps. La figura 5.21 [14] muestra los resultados obtenidos. Estos resultados validan los resultados teóricos obtenidos y demuestran experimentalmente la reducción del número de *bytes* de difusión utilizado por TFE.

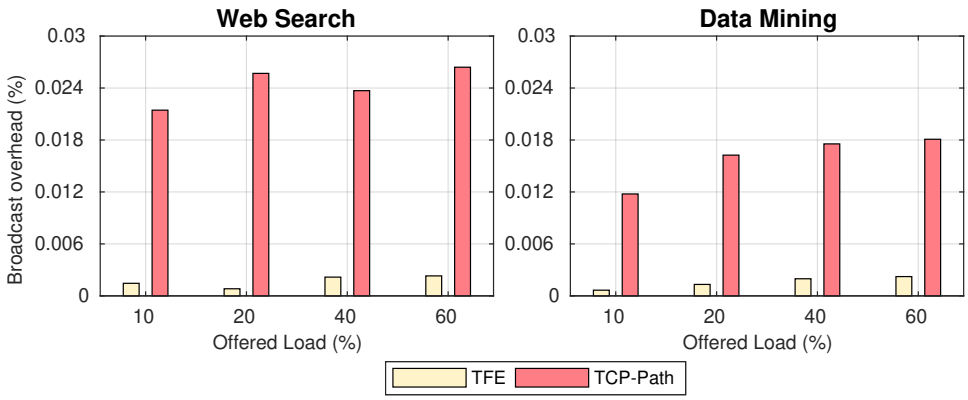


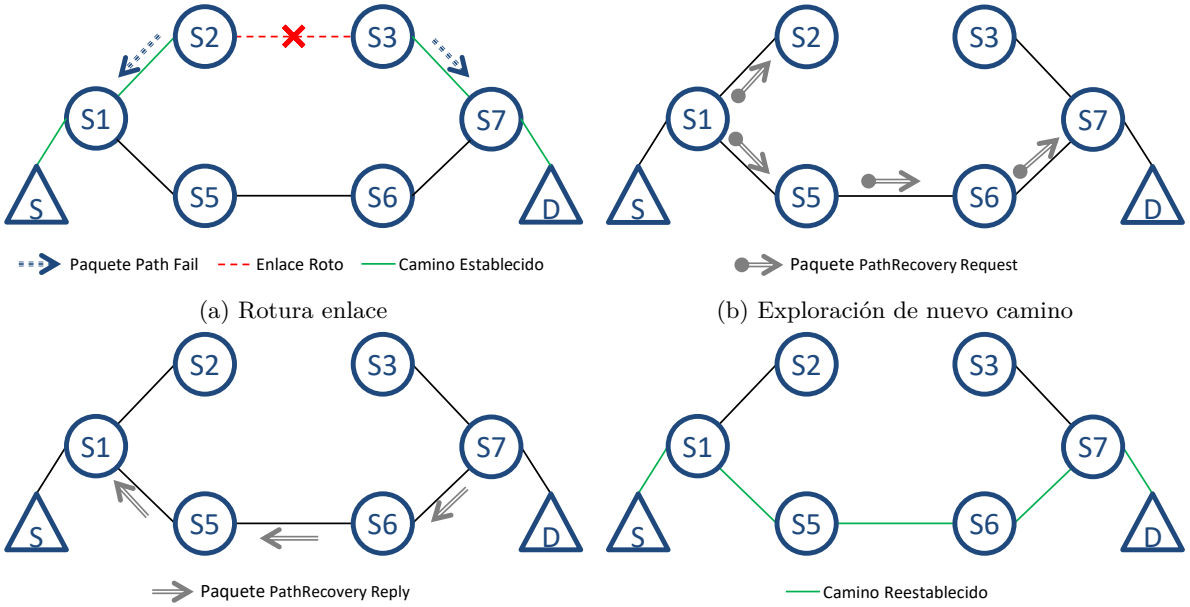
Figura 5.21: Porcentaje de broadcast generado por los protocolos de exploración en la topología Spine-Leaf 4x4 a 1 Gbps

5.3 Mecanismo de recuperación

El encaminamiento de flujos depende de manera directa del estado de la red en cada momento. Es por esto por lo que todo protocolo de encaminamiento de flujos debe tener la capacidad de recuperarse ante un eventual fallo de la red. En este trabajo ha propuesto dos protocolos de exploración de caminos y encaminamiento de flujos a nivel 2, TCP-Path y TFE, por lo tanto, siguiendo la premisa previa, estos protocolos deben tener la capacidad de regenerar los caminos dañados al producirse un fallo. Se propone, por ello, el siguiente mecanismo de recuperación distribuido.

Para explicar el proceso necesario para llevar a cabo este mecanismo se va a utilizar el ejemplo ilustrado en la figura 5.22. Sin perder de vista que este mecanismo solo actúa ante un eventual fallo de la red, el ejemplo toma como punto de partida la existencia de un flujo establecido entre los equipos finales *S* y *D*. En un instante cualquiera, la red sufre un fallo teniendo como consecuencia la caída del enlace que une los switches *S2* y *L3* perteneciente al camino utilizado para encaminar dicho flujo (ver figura 5.22a). En el instante en que los switches *S2* y *S3* detectan el fallo, deben generar una trama especial, *Path Fail*, con destino los switches origen y final del flujo. Para conocer si el switch que recibe la trama *Path Fail*, es uno de los dos switches (origen o final) cada switch debe comprobar que las direcciones origen o destino del flujo están dentro de su tabla de equipos finales conectados directamente a él. El resto de los switches, o switches intermedios, que reciban la trama *Path Fail*, debe buscar el puerto para reenviar la trama hacia el switch origen o destino del flujo, mediante una búsqueda en sus tablas de la entrada perteneciente al flujo definida por *dirección MAC origen, dirección MAC Destino, puerto TCP origen y puerto TCP Destino*. A continuación, una vez hayan encontrado el puerto, eliminarán dicha entrada, para, finalmente, enviar la trama hacia los switches frontera del flujo.

El switch origen del flujo una vez recibe la trama *Path Fail* crea una trama especial de difusión, *Path Recovery Request*. Esta trama será difundida por la red hasta llegar al switch frontera destino del flujo (ver figura 5.22b), generando un nuevo árbol de difusión



(a) Rotura enlace

(b) Exploración de nuevo camino

(c) Confirmación de camino recuperado

(d) Camino reestablecido

Figura 5.22: Proceso de recuperación

con origen y destino los switches origen y destino del flujo. Para evitar bloqueos se aplica el mismo mecanismo que en el proceso de exploración explicado en TCP-Path. Los switches intermedios que reciben esta trama especial actualizarán sus registros para apuntar en ellos el nuevo puerto.

Para confirmar el nuevo camino, el switch frontera destino debe generar una nueva trama *Path Recovery Reply*. A diferencia de la trama anterior, se trata de una trama unicast que se encamina por el camino explorado por la trama *Path Recovery Request*. Cada uno de los switches que reciban esta trama debe confirmar la entrada generada por *Path Recovery Request* e introducir una nueva en sentido contrario (ver figura 5.22c). Este proceso termina cuando la trama *Path Recovery Reply* es recibida por el switch frontera origen (ver figura 5.22d).

5.4 Conclusiones

En esta sección se han presentado dos nuevos protocolos: *TCP-Path* y, su variante, *TFE*. TCP-Path es un protocolo especialmente diseñado para flujos TCP y con la capacidad de generar caminos bajo demanda haciendo uso de mecanismos de exploración por difusión de tramas, mientras que, TFE es una variante del protocolo anterior que surge como solución para los problemas de escalabilidad que tiene TCP-Path. También se ha presentado un mecanismo que evita la generación de caminos con rebotes. Este mecanismo evita que los flujos cambien de capa dentro de las redes jerárquicas de manera descontrolada.

El protocolo TCP-Path utiliza, para la generación de los caminos, la latencia como parámetro de búsqueda, es decir, busca el camino más rápido entre un origen y un destino en el momento en que un flujo TCP necesita ser encaminado. Mientras que, TFE aplica esta lógica, únicamente, a los flujos de tipo *Elefante*.

Tras las pruebas realizadas y como muestran los resultados, tanto TCP-Path como TFE son capaces de encaminar el tráfico TCP a través de la red estableciendo los distintos caminos de forma correcta. Además, los resultados muestran que TCP-Path tiene un rendimiento superior a ECMP tanto en BW como en FCT. Esta mejora se debe a que TCP-Path es capaz de realizar un reparto de la carga de la red más eficiente, debido a que aprovecha mejor los recursos evitando en gran medida la colisión de flujos de tipo *Elefante*. Del mismo modo, las pruebas han demostrado que el rendimiento obtenido por TFE es ligeramente inferior al de TCP-Path pero superior a ECMP. Sin embargo, las pruebas también muestran como TFE tiene una escalabilidad mayor que la ofrecida por TCP-Path. Esto se debe a que tanto el tráfico de difusión que necesita para funcionar como el número de entradas en tabla son mucho menores con respecto a TCP-Path.

Por último, en este capítulo se ha presentado, también, un mecanismo de recuperación para los protocolos TCP-Path y TFE. Es bien sabido por la comunidad, la importancia de este tipo de mecanismos para garantizar la resiliencia de la red ante eventuales fallos. Se trata de un proceso sencillo, rápido y eficiente que aprovecha la versatilidad de las exploraciones de red para generar nuevos caminos entre origen y destino, manteniendo como parámetro de calidad la latencia del camino y conservando la bidireccionalidad de los caminos restaurados.

Capítulo 6

Descubrimiento de caminos y topologías en redes con dispositivos híbridos SDN

La nueva arquitectura SDN propone realizar una modificación sobre el modelo arquitectónico tradicional, eliminando la 'inteligencia' disponible en cada uno de los nodos que conforman la capa de infraestructura, y entregando dicha 'inteligencia' al *Plano de Control*. Para ello, el *Plano de Control* necesita conocer el estado de cada uno de los switches SDN mediante un intercambio de paquetes de control con ellos. Posteriormente, debe definir qué camino debe seguir cada flujo instalando una serie de reglas en cada uno de los dispositivos de la *Capa de Infraestructura*. A veces, incluso, modificar los servicios ofrecidos por la red mediante el manejo de las Network Function Virtualizations (NFVs).

Para poder realizar las tareas anteriormente mencionadas, la *Capa de Control* necesita manejar una serie de protocolos capaces de obtener la información topológica necesaria. Como es de esperar, no existe un único protocolo capaz de recolectar información topológica en redes, sino que existen multitud de protocolos destinados a recopilar únicamente un pequeño fragmento de toda la información necesaria. Entre todos ellos, esta tesis desea destacar a los protocolos encargados de obtener la información acerca de la topología de la red. Actualmente, el protocolo establecido como estándar para realizar esta labor es el protocolo OFDP [10], el cual está basado en el protocolo LLDP [61].

En este capítulo se propone TEDP [13], un protocolo cuyo objetivo es recolectar y transmitir al *Plano de Control* la información referente a la topológica de la red SDN. Para ello, este nuevo protocolo utiliza la exploración de la red como mecanismo para obtener dicha información, pero a su vez, es capaz de recolectar y proporcionar, sin un procesamiento extra por parte del controlador, un árbol de expansión entre los nodos de la red, gracias al cual el controlador puede ser capaz de encaminar el tráfico dentro de la red. Asimismo, se ha realizado el desarrollo de dos implementaciones diferentes para este protocolo. Primero, una *centralizada*, donde el *Plano de Control* es la única entidad

que conoce los diferentes pasos para llevar a cabo el protocolo y, por lo tanto, dejando únicamente la función de reenvío a los switches SDN. Luego, otra *híbrida*, donde parte de la lógica establecida en el *Plano de Control* para realizar el protocolo se lleva a cabo en los switches híbridos, haciendo que el *Plano de Control* únicamente se encargue de iniciar el proceso de exploración y del análisis de los datos recibidos, siguiendo la misma filosofía usada para el mecanismo de recuperación híbrido presentado en el capítulo 4.

6.1 TEDP: Tree Exploration Discovery Protocol

TEDP se ha definido como un protocolo capaz de descubrir los enlaces que conectan cada uno de los switches que conforman la red. Por este motivo, se han fijado como objetivos para el diseño de este nuevo protocolo la obtención de toda la información necesaria para que el *Plano de Control* sea capaz de generar un grafo que describa de forma correcta la topología de la red existente y la obtención de la información suficiente para que el *Plano de Control* pueda establecer un árbol de expansión con raíz un switch cualquiera de la red. Además, el desarrollo de este protocolo deberá utilizar como máximo el mismo número de paquetes que el utilizado por la implementación centralizada del protocolo de referencia OFDP.

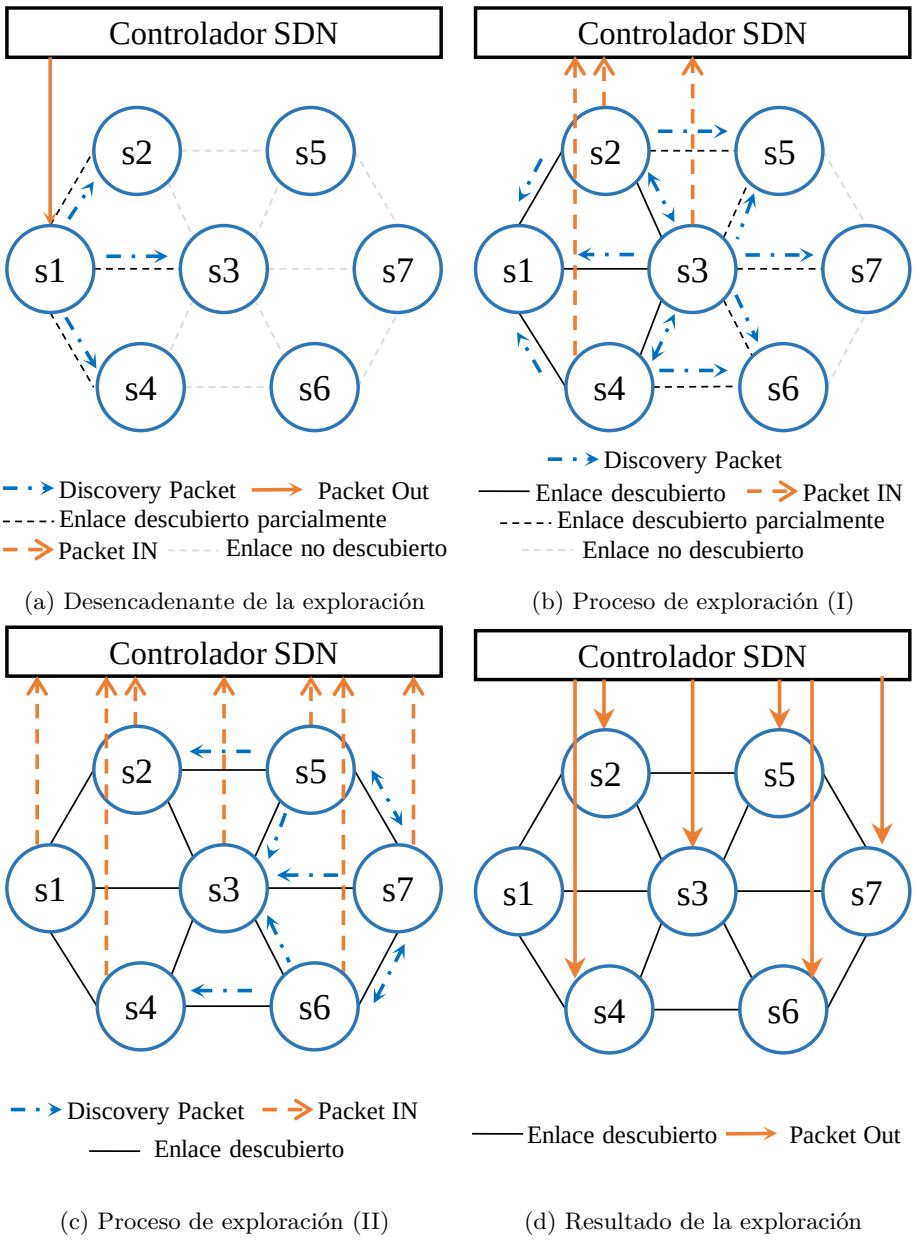
6.1.1 Funcionamiento básico de TEDP

Para explicar el funcionamiento del protocolo propuesto se va a hacer uso del ejemplo representado por la figura 6.1 formado por un controlador SDN y siete switches conectados entre ellos según dicha figura.

En un instante de tiempo, elegido de forma aleatoria, el proceso de TEDP es iniciado por el controlador, para ello, crea una trama especial de descubrimiento (*Path Discovery*). A continuación, selecciona a uno de los siete switches para iniciar el proceso de exploración. En el caso representado en la figura, el controlador ha elegido al switch *S1* para iniciar el proceso de exploración. Finalmente, el controlador encapsula la trama de exploración en un *Packet_Out* y, posteriormente, se lo hace llegar al switch previamente elegido. Este proceso se muestra en la figura 6.1a.

Una vez, el switch *S1* ha recibido el paquete *Packet_Out*, representado por la flecha de color naranja dentro de la figura 6.1a, obtiene la trama *Path Discovery* original desencapsulando el *Packet_Out* recibido. El switch, una vez finaliza el desencapsulado, reenvía la trama *Path Discovery* por todos sus puertos menos el puerto de conexión con el controlador. Este hecho establece el inicio del proceso de exploración.

Cada uno de los switches vecinos (en el caso del ejemplo S2, S3 y S4), una vez reciban la trama *Path Discovery* proceden a realizar dos procesos paralelos. Por un lado, comprueban si la trama *Path Discovery* recibida debe ser reenviada a todos sus vecinos o, por el



(a) Desencadenante de la exploración

(b) Proceso de exploración (I)

(c) Proceso de exploración (II)

(d) Resultado de la exploración

Figura 6.1: Proceso llevado a cabo por el protocolo TEDP

contrario, debe ser descartada para evitar bucles en la red. Para ello, cada switch utiliza un mecanismo basado en el mecanismo para la prevención de bucles establecido en el protocolo *ARP-Path*. En el resto de los casos, la trama *Path Discovery* es descartada (ver figura 6.1b). Por otro lado, el switch debe encapsular una copia de la trama *Path Discovery* recibida en un paquete *Packet_In* para hacer llegar la información del enlace descubierto al controlador. En el instante representado en la figura 6.1b, el controlador gracias a la información recibida a través de las tramas *Path Discovery* enviadas por los switches S2, S3 y S4 y al intercambio de paquetes *Hello* propio del protocolo *OpenFlow* es capaz de crear un grafo compuesto por siete switches SDN (S1, ..., S7) y tres enlaces unidireccionales. Estos enlaces unen S1 con S2, S1 con S3 y S1 con S4.

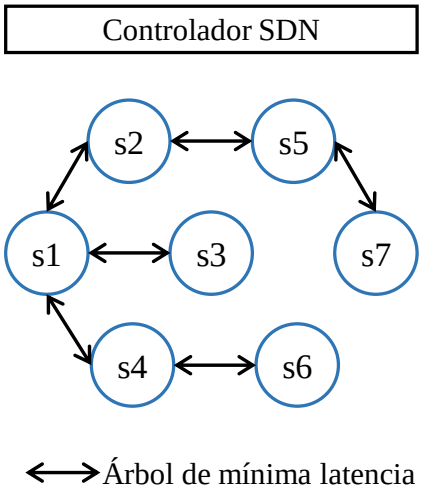


Figura 6.2: Árbol de encaminamiento descubierto

En la figura 6.1c, se observa cómo todos los switches de la red que reciben la trama *Path Discovery* repiten el proceso explicado en el párrafo anterior, es decir envían la copia de la trama *Path Discovery* al controlador y, en el caso correspondiente, la reenvían a todos sus vecinos. Los switches envían al controlador tantas tramas *Path Discovery* encapsuladas en paquetes *Packet_In* como puertos activos tengan. Este hecho se debe a que cada una de las tramas *Path Discovery* contiene, únicamente, la información relativa a un enlace unidireccional de la topología. Por lo que, para que el controlador sea capaz de descubrir todos los enlaces de la topología necesita recibir al menos una trama *Path Discovery* por cada puerto existente en la red. El proceso de descubrimiento finaliza en el instante en que todos los switches que componen la red han propagado una única vez, la trama *Path Discovery* (ver figura 6.1d).

El protocolo propuesto finaliza una vez el controlador dispone de toda la información topológica recibida a través de las diferentes tramas *Path Discovery*. Primero, tiene la información de todos los nodos, así como todos los enlaces que los interconectan. Y segundo, un árbol de expansión con raíz el switch seleccionado al inicio de la exploración ya que el proceso de exploración da los caminos más rápidos a cada nodo. Dicho árbol se obtiene directamente de la información procedente de la primera trama *Path Discovery* recibida de cada uno de los switches. Un ejemplo de un árbol de expansión con raíz el switch *S1* ha sido representado en la figura 6.2.

6.1.2 Diferencias de funcionamiento entre modo centralizado y modo híbrido

Como se ha expuesto al comienzo del capítulo, este protocolo de descubrimiento dispone de dos modos de funcionamientos diferentes, uno totalmente centralizado y otro híbrido, y aunque, el funcionamiento básico del protocolo TEDP es el mismo independientemente del modo utilizado, existen algunas diferencias entre ambos.

La primera diferencia entre los dos modos se da en el mismo momento en que el protocolo entra en funcionamiento por primera vez, ya que en el modo centralizado, el controlador necesita instalar, previamente al inicio del protocolo, dos reglas en cada uno de los switches. La primera para habilitar el envío de la trama *Path Discovery* al controlador y la segunda para habilitar la difusión de la trama por todos los puertos activos menos por él de entrada. En cambio, en el modo híbrido no es necesario que el controlador instale dichas reglas para que el protocolo propuesto funcione correctamente.

La segunda diferencia radica en qué entidad ejecuta el mecanismo para evitar la aparición de bucles en la red. Mientras que en el modo de funcionamiento centralizado, el controlador debe evitar la aparición de bucles controlando la difusión de las tramas de descubrimiento en todo momento. Por este motivo, en este modo de funcionamiento el controlador es el encargado de aplicar el mecanismo de prevención de bucles similar al de ARP-Path. Para ello, en el momento que un switch le hace llegar una trama *Path Discovery*, el controlador elimina la regla encargada de la propagación de la trama de descubrimiento mediante el envío de un paquete *Flow_Mod*. En el modo de funcionamiento híbrido son los propios switches los que aplican el mecanismo para evitar bucles sin necesidad de intervención por parte del controlador.

La última diferencia está relacionada con el envío de la información al controlador. En modo centralizado, el controlador una vez ha recibido tantas tramas *Path Discovery* como puertos tiene el switch SDN que se las ha enviado, envía un paquete *Flow_Mod* para eliminar la regla de reenvío de tramas *Path Discovery* al controlador. Mientras que, en el caso del modo híbrido, son los propios switches quienes toman la decisión de dejar de enviar tramas al controlador, sin necesidad de que el controlador les comunique nada. Esta acción se realiza con el objetivo de reducir el número de tramas enviadas al controlador al mínimo posible.

6.2 Paquete de descubrimiento Path Discovery

En este capítulo se propone TEDP, un nuevo protocolo descubrimiento basado en la exploración de la red mediante difusión. Este protocolo necesita de algún elemento capaz de recolectar información acerca de la red. Por este motivo, se ha definido una nueva trama llamada *Path Discovery*. Esta trama está formada por una cabecera Ethernet, compuesta por una dirección de difusión como dirección MAC destino, la dirección MAC principal del switch que reenvía la trama como dirección MAC origen y un *EtherType* definido para TEDP, la dirección MAC principal del switch seleccionado por el controlador para empezar el descubrimiento y, por último, el puerto por donde el switch va a reenviar la trama. La estructura de esta trama se muestra en la figura 6.3.

En dicha figura se realiza una comparación entre la trama utilizada por la implementación del protocolo OFDP incluida dentro del controlador RYU [92] y la trama

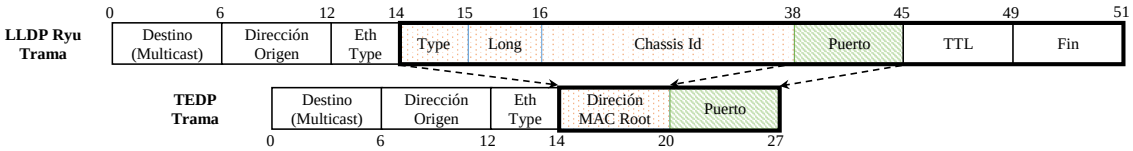


Figura 6.3: Comparación trama LLDP vs TEDP

utilizada por el protocolo propuesto. En ella se observa como la trama *Path Discovery* utilizada por TEDP es significamente más pequeña que la trama del protocolo LLDP, lo que garantiza que sea más sencilla y rápida de tratar por los switches. También se observa como la trama propuesta cumple con el tamaño mínimo exigido para propagarse por las redes Ethernet.

6.3 Diseño e implementación del protocolo TEDP

Como se ha mencionado en secciones anteriores, se han realizado dos implementaciones diferentes, la primera de ellas basada únicamente en controlador y desarrollada como una aplicación para dicho elemento. Mientras que, la segunda implementación se desarrolla de forma híbrida, lo que provoca la necesidad de realizar dos desarrollos diferentes. El primero consiste en desarrollar la lógica del protocolo TEDP dentro del software de un switch, mientras que el segundo consiste en implementar una aplicación para el controlador.

6.3.1 Diseño e implementación como aplicación SDN

La primera implementación de TEDP se realiza mediante el desarrollo de una aplicación SDN para el controlador *RYU*. Para realizar su implementación se ha establecido un diseño segmentado en tres grandes bloques: el primer bloque define las funciones que se deben realizar nada más iniciar la aplicación TEDP en el controlador; el segundo bloque, define el proceso que se realiza en cada uno de los nodos SDN que componen la red; y el tercer bloque, define las acciones que debe realizar el controlador ante un evento *Packet_In* generado por una trama *Path Discovery*. Estos tres bloques están representados en la figura 6.4 [13].

El primer bloque (ver figura 6.4a) comienza en el momento en que el controlador inicia la aplicación. El controlador al iniciar obtiene las características hardware de cada uno de los switches que componen la red. Para ello, envía un paquete *Features_Request* propio del protocolo *OpenFlow* capaz de solicitar las características técnicas de cada uno de los switches . Una vez los switches proporcionen la información solicitada. El controlador almacena en una tabla el identificador (DpId) de cada switch junto a la lista de direcciones MAC de cada uno de sus puertos. Esta estructura será de gran importancia más adelante ya que gracias a ella el controlador sabrá qué puertos han enviado o no su información topológica.

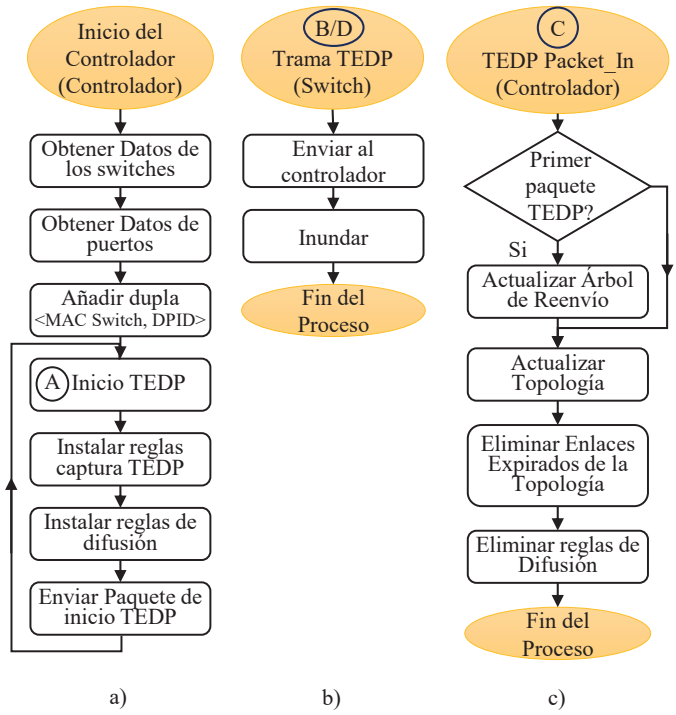


Figura 6.4: Diagrama de flujo de la lógica TEDP sobre controlador SDN [13]

Tras esto el controlador inicia el protocolo *TEDP* con la configuración de las reglas de reenvío necesarias para que los switches sean capaces de reenviar los paquetes *Path Discovery* tanto al controlador como a todos sus vecinos. Esta acción se realiza mediante el envío de paquetes *Flow_Mod* a todos los switches de la red.

Antes de iniciar el proceso de descubrimiento, el controlador debe seleccionar cual será el nodo raíz del futuro árbol de expansión, para ello selecciona un switch de entre todos los que componen la red. Posteriormente, le hace llegar una trama *Path Discovery* encapsulada en un paquete *Packet_Out* para dar comienzo al proceso de descubrimiento. A partir de este momento, todas las acciones que el protocolo deba realizar van a ser desencadenadas mediante eventos producidos por la llegada de una trama *Path Discovery* ya sea a uno de los switches (ver figura 6.4b) o al propio controlador (ver figura 6.4c).

El proceso que debe seguir un switch para llevar a cabo el protocolo TEDP en su versión centralizada se muestra en la figura 6.4b. El switch únicamente debe ejecutar las reglas instaladas por el controlador en su tabla de flujos. Estas reglas son el envío de una copia de la trama *Path Discovery* recibida al controlador mediante su encapsulación en un paquete *Packet_In* y la difusión de la trama por todos los puertos incluidos el de entrada.

Del mismo modo, el mecanismo propuesto para el controlador se muestra en la figura 6.4c. Este proceso comienza justo cuando una trama *Path Discovery* es encapsulada en un paquete *Packet_In* llega al controlador. El controlador, una vez obtenida la trama original, comprueba si el paquete *Path Discovery* recibido es el primer paquete enviado por el switch. En caso afirmativo, el controlador actualiza el árbol de expansión. A

continuación, actualiza el grafo topológico, para ello, introduce o actualiza la información referente al enlace recibido. Para finalizar el proceso, el controlador envía al switch un paquete *Flow_Mod* para eliminar la regla encargada de difundir la trama por todos los puertos del switch y, además, comprueba si el switch que le envió el paquete *Packet_In* le ha enviado ya la información topológica de cada uno de sus puertos para, en tal caso, enviarle un paquete *Flow_Mod* para eliminar la regla encargada de reenviar las tramas *Path Discovery* al controlador.

6.3.2 Diseño e implementación como sistema híbrido SDN/distribuido

La segunda implementación realizada para el nuevo protocolo de descubrimiento es la implementación híbrida. En esta ocasión tanto el controlador como los switches son parte activa del protocolo, esto se debe a que parte de las funciones que antes se encontraban en el controlador ahora pasan a situarse en los switches. Este hecho deja al controlador exclusivamente para recopilar y analizar los datos obtenidos a través de las tramas *Path Discovery* recibidas. Por su parte, los switches son los encargados de llevar a cabo la exploración de la red. Como se ha dicho anteriormente, este modo híbrido sigue la filosofía del modo de recuperación híbrida presentado en el capítulo 4.

Al igual que en el caso centralizado (ver sección 6.3.1) el diseño híbrido se divide en tres bloques distintos, representados todos ellos en la figura 6.5. El primero, representado en la figura 6.5a, se encarga de la inicialización del protocolo en los switches. El segundo bloque, representado en la figura 6.5b, define el proceso que deben realizar los switches una vez hayan recibido una trama de exploración. Por último, la figura 6.5c muestra la lógica que debe seguir el controlador para procesar correctamente la información recibida. En este caso, los bloques 2 y 3 se desencadenan mediante eventos generados por la entrada de una trama *Path Discovery*, mientras que el bloque 1 se desencadena de manera aleatoria.

Al iniciar la red, cada uno de los switches arrancan un proceso que decide de manera aleatoria si se debe o no iniciar el protocolo de descubrimiento. En el caso de que el switch decida iniciar el proceso de exploración, debe difundir una trama de descubrimiento *Path Discovery* por todos sus puertos disponibles excepto por el puerto que le conecta con el controlador. Este proceso se muestra en la figura 6.5a. En el momento que una trama *Path Discovery* ha sido recibida por uno de los switches que conforman la red, se pasa a ejecutar el proceso representado por la figura 6.5b. Cada switch envía una copia del mensaje recibido al controlador mediante un paquete *Packet_In*. De forma paralela, difunde la trama recibida por cada uno de sus puertos, modificando antes de su envío el campo 'puerto' de la trama. Dicho campo tomará el valor del puerto por el que la trama va a ser enviada. Para finalizar el mecanismo y evitar futuros bucles, el nodo apunta el puerto de entrada de la trama original *Path Discovery* junto a la dirección física del switch que inició el proceso de descubrimiento y una marca temporal. La dirección física del nodo que inició la exploración está indicada en el campo 'dirección MAC root'.

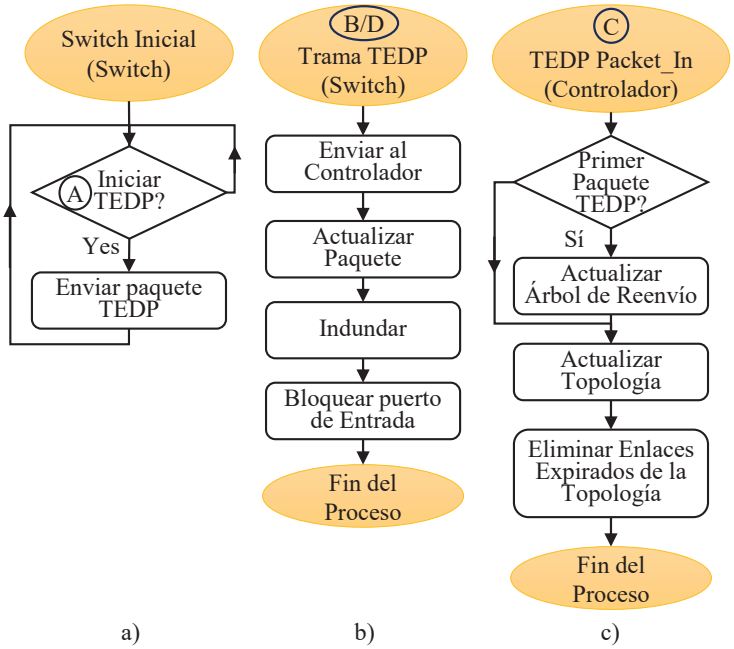
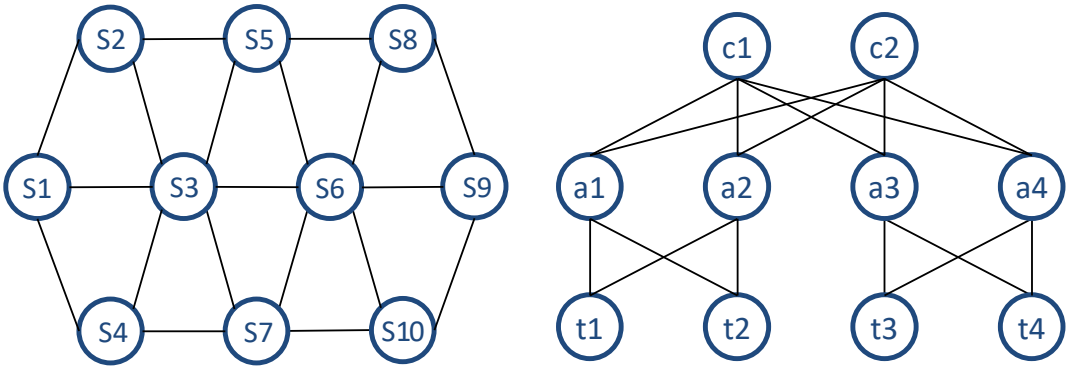


Figura 6.5: Diagrama de flujo de la lógica TEDP sobre sistema híbrido [13]

El controlador, al igual que los switches de la red, ejecuta su proceso, cuando una trama *Path Discovery* encapsulada en un paquete *Packet_In* llegue a él. En ese momento el controlador realiza las acciones representadas en la imagen 6.5c. El controlador, tras obtener la trama *Path Discovery* original, comprueba si ha recibido anteriormente una trama *Path Discovery* procedente del mismo switch. En caso de no haber recibido ninguna otra trama, el controlador pasa a actualizar el árbol de expansión con raíz el switch que inició el descubrimiento. Al igual que en el caso expuesto en el párrafo anterior, esta información viene en el campo 'dirección MAC root' de la trama. A continuación, actualiza la topología general, ya sea insertando un nuevo enlace o actualizando la marca temporal del enlace. Para crear un nuevo enlace dentro del grafo topológico, el controlador toma el valor del campo *puerto* de la trama, el puerto de entrada del switch que envía el *Packet_In* y los DpIds de los switches origen y destino del enlace.

6.4 Evaluación

En este capítulo se presenta la evaluación del protocolo TEDP, cuyo objetivo es el descubrimiento de la topología utilizando la exploración como mecanismo de descubrimiento. Para evaluar el mecanismo se ha decidido seguir la misma filosofía utilizada para la evaluación del protocolo de recuperación híbrida propuesto en el sección 4.5, es decir, un primer conjunto de pruebas en que se comprueba el correcto funcionamiento del protocolo y, un segundo conjunto de pruebas en que se evalúa y compara el rendimiento de este protocolo con el mecanismo habitualmente en la arquitectura .



(a) Topología formada por 10 Nodos (b) Topología *VL2*

Figura 6.6: Topologías formadas por 10 nodos [13]

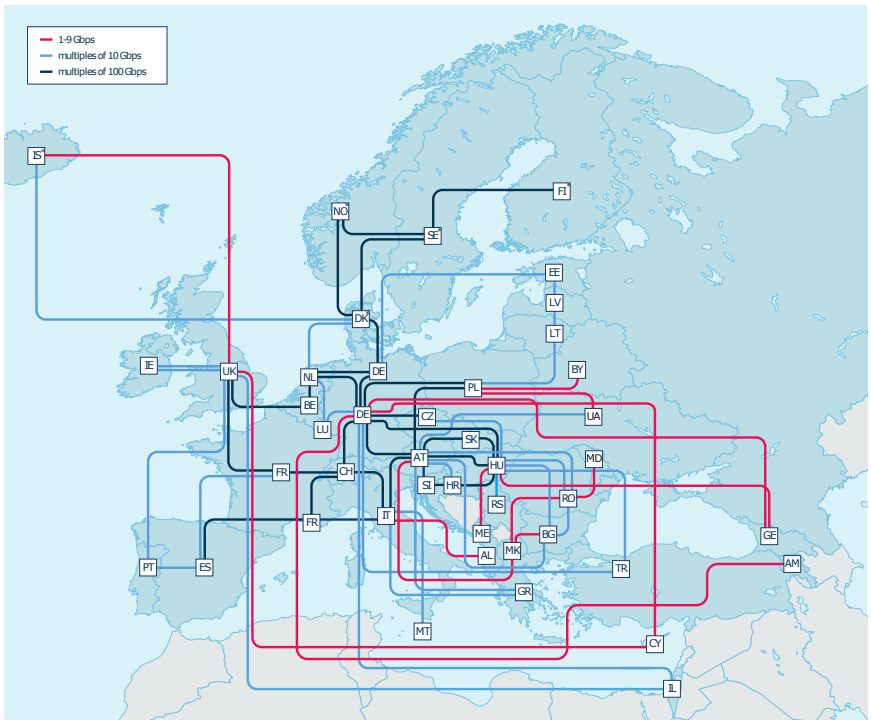
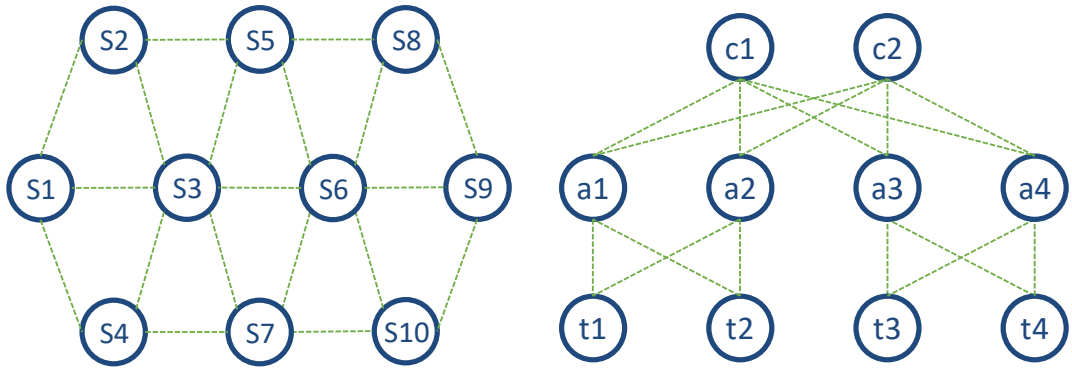


Figura 6.7: Topología *GEANTS* [93]

6.4.1 Entorno de pruebas

Para realizar los dos conjuntos de pruebas definidos en la sección anterior se ha hecho uso de diferentes topologías de red. Para las pruebas en que se pretende comprobar el correcto funcionamiento del protocolo se ha utilizado una topología sencilla formada por 10 nodos, la cual se puede observar en la figura 6.6a y una topología jerárquica, en este caso la *VL2*, representada en la figura 6.6b) y formada por 2 *Cores*, 4 *Agregados* y 4 *ToRs*. Para las pruebas en que se pretende evaluar el rendimiento del protocolo se ha utilizado una topología real de gran tamaño conocida como *GÉANT* Paneuropea [93] formada por 44 nodos, todos ellos distribuidos a lo largo del continente europeo, y 144 enlaces entre ellos. Dicha topología se puede observar en la figura 6.7 [93]



(a) Grafo topológico formado en el controlador para topología de 10 nodos

(b) Grafo topológico formado en el controlador para topología *VL2* de 10 nodos

Figura 6.8: Grafos topológicos

Por último, para la realización de las pruebas se ha contado con una infraestructura formada por un ordenador con procesador Intel(R) Core(TM) i7 y 24 GBytes de Memoria RAM. Asimismo, se ha utilizado Mininet [82] como herramienta de emulación, con *Cubic* como control de congestión. El desarrollo de la lógica de controlador se ha realizado en una aplicación basada en el controlador *RYU* e implementada en *Python*.

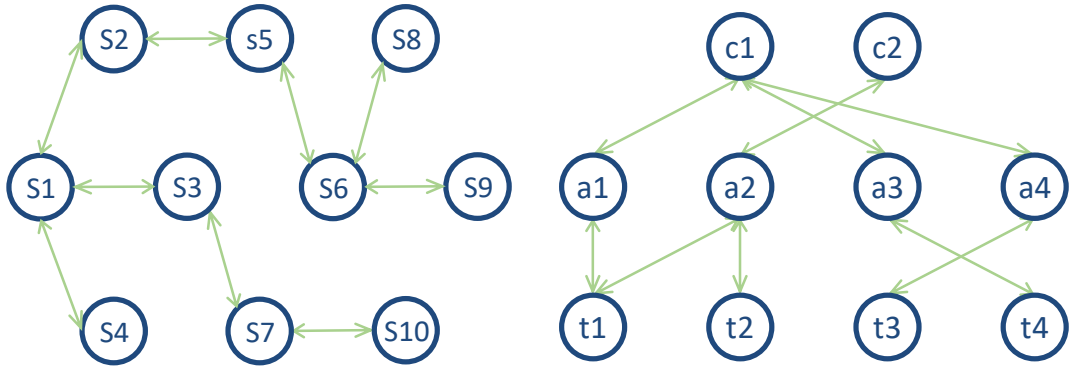
6.4.2 Prueba de concepto

Esta prueba se ha realizado usando las topologías indicadas en la subsección anterior y sin tráfico de fondo puesto que solo se quería comprobar el correcto funcionamiento del protocolo. Además, la prueba se ha sido repetida 10 veces para cada una de las topologías seleccionado en cada una de las repeticiones un switch aleatorio como punto de inicio de la exploración del protocolo.

Los resultados obtenidos se muestran en las figuras 6.8a y 6.8b. En ellas se observa como el controlador es capaz de formar en su base de datos un grafo que describe correctamente la topología de ambas redes (topología formada por los 10 nodos y topología *VL2*). Del mismo modo, los resultados muestran que el controlador, tras haber elegido como switch de inicio de la exploración el nodo *S1* para las pruebas realizadas sobre la topología formada por los 10 nodos y el nodo *t1* para el caso de la topología *VL2*, es capaz de generar correctamente un árbol que conecta todo los switches y cuya raíz es el switch elegido para iniciar el protocolo. Estos resultados se muestran en las figuras 6.9a y 6.9b.

6.4.3 Pruebas de rendimiento

Con este conjunto de pruebas se pretende cuantificar la mejora de rendimiento existente entre el protocolo propuesto y el protocolo más utilizado actualmente, *OFDP*. Para realizar esta evaluación se ha decidido utilizar como topología la topología *GEÀNT*, más concretamente la representada en la figura 6.7. Utilizando esta topología se han realizado



(a) Árbol topológico para topología de 10 nodos con raíz en s1

(b) Árbol topológico para topología *VL2* de 10 nodos con raíz en t1

Figura 6.9: Arboles topológicos

dos estudios diferentes. Primero, un estudio teórico donde se desea obtener el número máximo de tramas que necesitan cada uno de los protocolos (tanto OFDP como TEDP) para generar el grafo topológico de forma correcta. Mientras que, el segundo es un estudio experimental, donde se mide el número de tramas utilizadas por cada uno de ellos para obtener el grafo topológico de la red en emulación.

Estudio teórico del número de paquetes necesario en TEDP para descubrir la red GÉANT

Los cálculos teóricos se basan en el funcionamiento de los dos protocolos a comparar. Ambos protocolos necesitan conocer las conexiones establecidas entre los diferentes nodos para generar el grafo de la topología. Centrando el estudio primero en OFDP, se sabe que es necesario instalar una regla en cada uno de los switches SDN mediante la cual se indica a los switches que cada vez que reciban una trama cuyo *EtherType* coincida con él del protocolo deben enviarla al controlador mediante un paquete *Packet_In*. Después, el controlador genera y envía tantas tramas LLDP encapsuladas en paquetes *Packet_Out* como puertos existen en la red. Una vez la trama LLDP llega al switch, es reenviada siguiendo las instrucciones incluidas en el *Packet_Out* por el controlador. Por último, cada switch que recibe la trama LLDP la envía al controlador encapsulándola en un paquete *Packet_In*. En resumen, este proceso necesita: un *Packet_Out*, una trama LLDP y un *Packet_In* por cada uno de los puertos que existan en la topología (ver ecuación 6.1).

Por su parte, TEDP necesita únicamente de un *Packet_Out* creado por el controlador para iniciar el proceso de descubrimiento y de una trama de descubrimiento por cada puerto que exista en la red (ver ecuación 6.1). Este hecho se debe a que el paquete creado en el controlador es un paquete de difusión con el que se pretende explorar todos y cada uno de los enlaces de la red. Además, requiere de un paquete *Packet_In* por cada puerto de la red para comunicarse con el controlador(ver ecuación 6.1), ya que cada vez que un switch recibe una trama de descubrimiento debe enviarla al controlador para que esté disponga de toda la información de la red.

Packet #	Plano de Control			Plano de Datos
	Packet_In	Packet_Out	Flow_Mod	
LLDP	144	144	44	144
TEDP-S	144	1	88	144
TEDP-H	144	0	0	144

Tabla 6.1: Resultados teóricos de LLDP and TEDP

Los resultados obtenidos se muestran en la tabla 6.1. En ellos se observa como el número de paquetes *Packet_In* y de exploración utilizados por ambos protocolos coincide, ya que ambos protocolos necesitan explorar cada uno de los enlaces existentes en la topología y notificar la información obtenida al controlador. Asimismo, el número de paquetes *Packet_Out* utilizado por el protocolo TEDP, tanto en su versión centralizada (TEDP-S) como en su versión híbrida (TEDP-H), es mucho menor que en OFDP. Esto se debe a que OFDP necesita que el controlador envíe tantos paquetes de descubrimiento como puertos existan en la topología, mientras que TEDP solo necesita enviar un único paquete a la red para su descubrimiento. Por último, también se observa como el número de paquetes necesarios para la instalación de las reglas por OFDP es menor que el número de paquetes utilizados por la implementación TEDP-S pero mucho mayor que los utilizados por la versión TEDP-H.

$$Num_{Packet} = \sum_{n=1}^{n=N_{Port}} n \tag{6.1}$$

Estudio experimental del número de paquetes necesarios en TEDP para descubrir la red GÉANT

Para validar los resultados teóricos, se han realizado un conjunto de pruebas experimentales sobre la misma topología. Estas pruebas se han realizado utilizando OFDP, *TEDP-S* y *TEDP-H*. Esta última implementación ha sido modificada para obtener el número de paquetes en el peor caso. Para ello, se ha eliminado la posibilidad de utilizar el conocimiento sobre el puerto de salida de la trama en cada uno de los enlaces y se ha incluido el reenvío de la trama de difusión también por el puerto de entrada. Como es sabido, esta implementación no necesita del reenvío de la trama por el puerto de entrada, ya que al conocer el puerto de salida del switch anterior, el controlador es capaz de conocer los dos sentidos del enlace. Estas pruebas se han repetido al menos 10 veces para obtener unos resultados cuyo nivel de confianza supere el 95 %. Los resultados obtenidos son los mostrados en la tabla 6.2 [94].

En dicha tabla se observa como los resultados experimentales obtenidos para OFDP y *TEDP-H* son muy aproximados a los resultados teóricos mostrados en la subsección anterior. Por su parte, la implementación *TEDP-S* muestra un número de paquetes algo superior al número de paquetes calculado en el estudio teórico. La diferencia entre el

Packet #		Plano de Control			Plano de Datos
		Packet_In	Packet_Out	Flow_Mod	
OFDP	$\overline{x}$	145,00	160.06	44,00	160.06
	σ	14,24	19,78	0,00	19,78
TEDP-S	$\overline{x}$	170,63	1,00	227,44	170,63
	σ	2,31	0,00	1,71	2,31
TEDP-H	$\overline{x}$	146,00	0,00	0,00	146,13
	σ	0,73	0,00	0,00	0,81

Tabla 6.2: Comparación experimental entre LLDP y TEDP

estudio teórico y el estudio experimental se debe al tiempo en que la regla de difusión de los paquetes de descubrimiento está activa sin necesidad. Este tiempo es el que el controlador utiliza para crear y enviar el paquete encargado de eliminar la regla más el tiempo que utiliza el switch para eliminar dicha regla. Por último, se puede observar que la implementación híbrida modificada (*TEDP-H*), es decir, una implementación donde no se utiliza el valor del campo 'puerto de salida' introducido por el switch anterior para reducir el número de paquetes de exploración, obtiene unos resultados prácticamente idénticos a los obtenidos en los cálculos teóricos.

6.5 Conclusiones

En este capítulo se ha presentado un nuevo protocolo de descubrimiento de topologías que hace uso de mecanismos de exploración para redes basadas en controlador como alternativa al protocolo OFDP.

El nuevo protocolo, denominado TEDP, se ha presentado mediante la realización de dos implementaciones distintas. Un primer desarrollo basado únicamente en controlador, especialmente diseñada para la arquitectura SDN/*OpenFlow* y una segunda aproximación especialmente diseñada para dispositivos híbridos SDN.

Tras las pruebas realizadas se ha observado como, por un lado, TEDP es capaz de facilitar la información necesaria para la creación del grafo topológico, así como del árbol de confluencia con raíz un nodo cualquiera de la red, el cual puede ser usado para encaminar tráfico y establecer caminos entre nodos. También han demostrado que el protocolo obtiene la información topológica haciendo uso de como máximo el mismo número de paquetes que OFDP. Por lo tanto, se puede concluir que TEDP es una alternativa viable y realista a OFDP en entornos SDN para la obtención de información topológica sobre la red con la ventaja de aportar información complementaria para encaminamiento y establecimiento de flujos en la red debido a los árboles de confluencia que se generan a partir de cada uno de los switches de la red.

Capítulo 7

Conclusiones y trabajos futuros

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se derivan del trabajo.

7.1 Conclusiones

Como se ha visto a lo largo de la tesis, las redes de datos están en continua evolución y, en muchos casos, aumentando su complejidad. Esta tesis pretende paliar algunos de los problemas que surgen tras el aumento del número de usuarios que las utilizan y de servicios que hacen uso de ellas, tales como caídas del servicio, disminución del rendimiento y complejidad en su administración a pesar del uso de nuevas arquitecturas de red como las redes SDN.

Esta tesis comienza proponiendo un mecanismo de recuperación híbrido como posible mejora para aumentar la robustez de las redes (capítulo 4). Este mecanismo utiliza el conocimiento acerca de la red que dispone el plano de control de la arquitectura SDN para desplegar un mecanismo de recuperación distribuido en switches híbridos SDN, minimizando así el impacto en el plano de control. Este mecanismo ha sido probado y puesto en comparación junto a otros dos mecanismos de recuperación, uno centralizado y otro distribuido. Los resultados han mostrado las ventajas que caracterizan al nuevo mecanismo.

Posteriormente, esta tesis ha desarrollado el protocolo TCP-Path y su variante TFE (capítulo 5) con el objetivo de mejorar el rendimiento de los switches tradicionales así como facilitar su integración en switches híbridos debido a su sencillez. Ambos están diseñados para establecer caminos para flujos TCP de una manera eficiente. Los resultados obtenidos han mostrado que se obtienen mejoras de rendimiento en términos de BW y FCT. TFE consigue una mayor escalabilidad a costa de una pequeña reducción de rendimiento respecto a TCP-Path debido a que el mecanismo propuesto en TCP-Path sólo se aplica a los flujos de mayor tamaño circulando por la red.

Finalmente, reutilizando el paradigma de exploración de caminos, se ha desarrollado un nuevo protocolo, TEDP (capítulo 6), para redes SDN, capaz de proporcionar la información topológica junto a un árbol de distribución capaz de encaminar el tráfico en la red descubierta. Esta última propuesta mejora el coste asociado al proceso de adquisición de la información y la propia información recopilada tras un proceso de descubrimiento iniciado por el plano de control SDN. Además, se dispone de un modo de funcionamiento híbrido que optimiza el proceso de exploración y recuperación de información requerido por TEDP.

Como se ha comprobado, aun teniendo todas las contribuciones una base común, la aplicación del paradigma de la exploración de caminos a los diferentes protocolos y mecanismos propuestos dentro del marco de tesis, estas propuestas mejoran características diferentes de las redes de comunicaciones. Por este motivo, se ha tomado la decisión de crear tres subsecciones diferentes.

7.1.1 Mecanismo de recuperación en entornos híbridos SDN/Distribuidos

El primero objetivo fijado para la realización de esta tesis establecía la necesidad de realizar un estudio, un desarrollo y una implementación de un nuevo mecanismo que utilizando las bondades que otorgan las diferentes arquitecturas de red existentes en la actualidad (SDN/Distribuido) fuera capaz de llevar a cabo una recuperación ante un fallo en la red. En el capítulo 4 se ha presentado un mecanismo de recuperación que intenta aprovechar las características principales de las arquitecturas actuales (arquitectura tradicional o distribuida y la arquitectura definida por software SDN). Este mecanismo hace uso del conocimiento que tiene de la red el plano de control SDN para localizar el destino de los flujos afectados por el error y de la exploración de caminos basada en difusión para localizar un nuevo camino entre el origen y el destino de cada uno de los flujos de forma eficiente.

Para comprobar el correcto funcionamiento del mecanismo de recuperación híbrido propuesto, así como su rendimiento. Se ha sometido al mecanismo de recuperación a una comparación con otros dos mecanismos de recuperación de naturaleza diferentes, un mecanismo de recuperación totalmente distribuido donde es la red la que mediante mecanismos de difusión y exploración consigue recuperarse tras un fallo, este mecanismo ha sido denominado como *Recuperación Distribuida*, y un mecanismo de recuperación que hace uso únicamente de la arquitectura SDN solicitando al plano de control un nuevo camino, es decir, una recuperación basada completamente en cómputo de caminos, este mecanismo ha sido denominado como *Recuperación Centralizada*.

Los resultados obtenidos en las pruebas han mostrado que el mecanismo funciona de forma correcta, encontrando un nuevo camino entre un origen y destino ante un fallo en la red en un tiempo inferior a 50 ms. Esta referencia temporal fue establecida por [87] como el tiempo máximo que debe tardar la red en recuperarse tras un fallo. Además, estos

resultados mostraron que el nuevo mecanismo reduce la cantidad de tiempo que la capa de control necesita ser utilizada por la red para realizar la recuperación, lo que permite que otra aplicación de alta prioridad puedan hacer uso del mismo. También mostraron que el tiempo de recuperación utilizado por el mecanismo propuesto cuando la carga de la red es media-baja ($\leq$ 70 %) es inferior al tiempo recomendado por [87] (50 ms) y aproximadamente igual al tiempo utilizado por la *Recuperación Distribuida* e inferior al necesario para llevar acabo la *Recuperación Centralizada*. Por el contrario, si la carga de la red es alta (>70 %) el tiempo consumido es superior al recomendado por [87] y no existe una diferencia apreciable entre las tres recuperaciones. Este hecho se debe al aumento del número de paquetes encolados a la entrada de cada uno de los switches, lo que provoca un aumento del tiempo que necesitan los mecanismos de exploración para encontrar una nueva ruta. Este problema se puede paliar utilizando colas con prioridad o aplicando mecanismos de Quality of Service (QoS) en los switches.

Con este trabajo se demuestra que el soporte de acciones híbridas en redes SDN es viable y, que puede ser una opción para descargar de tareas al plano de control en el caso de que se considere necesario como, por ejemplo, evitar sobrecarga o aumentar la escalabilidad del sistema.

Por último y, tras la revisión realizada a los resultados, se da por conseguido el objetivo fijado sobre el estudio e implementación de un nuevo mecanismo de recuperación en redes híbridas SDN/Distribuidas ya que, como han demostrado los resultados, el mecanismo de recuperación propuesto es capaz de restablecer la comunicación entre dos equipos finales y, además, el tiempo que necesita para realizar la recuperación es inferior a 50 ms cuando la carga de la red no supera el 70 % de su capacidad.

7.1.2 Protocolos de descubrimiento de caminos TCP-Path y TFE

Como segundo objetivo, en esta tesis se decidió plantear el desarrollo e implementación de un nuevo mecanismo capaz de seleccionar caminos de mínima latencia utilizando para ello mecanismos de exploración de red mediante difusión. Además, dicho mecanismo debía estar pensando especialmente para redes de arquitectura distribuida con alto rendimiento tales como redes de centros de datos.

En el capítulo 5 se han presentado *TCP-Path* y su variante TFE especialmente diseñados para el descubrimiento de caminos basados en exploración de la red para flujos TCP. TCP-Path descubre y establece un nuevo camino para cada nuevo flujo TCP que atraviese la red. Mientras que TFE establece un nuevo camino mediante descubrimiento únicamente para los flujos del tipo *Elefante* que recorren la red. Ambos protocolos seleccionan el camino de menor latencia que existe entre un origen y un destino dados utilizando mecanismos de exploración. En este capítulo también se ha presentado un mecanismo de recuperación aplicable para ambos. Se trata de un mecanismo sencillo, rápido y eficiente que aprovecha la versatilidad de las exploraciones de red para generar

nuevos caminos entre un origen y destino dados mantenido como parámetro de calidad la latencia del camino y, conservando a su vez, la bidireccionalidad de los caminos restaurados.

Ambas propuestas fueron sometidas a diferentes tipos de pruebas para comprobar su rendimiento en términos de BW y FCT. Estas pruebas se realizaron también con el protocolo de referencia actual para el encaminamiento de los flujos en capa de enlace (ECMP). Además, también se estudió su escalabilidad en términos de número de entradas en tabla y de tramas de difusión necesarios para la obtención de los diferentes caminos, tanto de forma analítica como experimental. Los resultados de las pruebas mostraron, en primer lugar, que ambos modos de funcionamiento son capaces de establecer caminos de forma correcta lo que corroboraba su objetivo básico. También mostraron que el rendimiento obtenido en términos de BW y FCT de *TCP-Path* es superior tanto al protocolo de referencia ECMP como al segundo protocolo propuesto TFE. Esto se debe TCP-Path permite descubrir cada uno de los caminos necesarios justo en el momento en que aparece cada uno de los flujos TCP en la red, lo que le garantiza un mejor balanceo de la carga debido a que tiene la información más actualizada del estado de la red justo en el momento de establecer cada uno de los caminos. Del mismo modo, los resultados también mostraron que TFE tiene un rendimiento ligeramente inferior a TCP-Path, pero superior al de ECMP, debido a que TCP-Path realiza un reparto de la carga más eficiente que TFE, ya que selecciona un camino para cada flujo TCP y, que TFE evita la colisión de flujos del tipo *Elefante* en mayor medida que ECMP.

También se ha realizado un segundo conjunto de pruebas con las que se buscaba estudiar la escalabilidad de ambas propuestas. Aunque TCP-Path mejora a ECMP, tiene un problema de escalabilidad debido a que necesita introducir una nueva regla en su tabla de aprendizaje por cada nuevo flujo en la red. Esto hace que, a medida que la carga de la red aumenta, el estado necesario en los diferentes nodos también aumenta. Por otro lado, también muestran como el tamaño de la red tiene una implicación directa en el número de paquetes de difusión que necesita el protocolo para su exploración completa. Por lo que a medida que aumenta el tamaño de la red, el porcentaje de tramas de difusión emitidas por el protocolo aumenta. Del mismo modo, los resultados muestran cómo el rendimiento de TFE es algo inferior al de TCP-Path, pero su escalabilidad es mucho mayor. Esto se debe a que a diferencia de él TFE solo plantea explorar un nuevo camino para un subconjunto pequeño del total de flujos TCP, lo que provoca que el estado (número de entradas en la tabla de encaminamiento) guardado por los nodos sea muy inferior al de TCP-Path y que el número de tramas de difusión utilizados por TFE también se reduzca considerablemente.

Como se ha observado los resultados obtenidos otorgan razones suficientes a esta tesis como para dar por conseguido el segundo objetivo referente al estudio e implementación de un nuevo protocolo de descubrimiento de caminos mediante exploración utilizando

como mecanismo para la exploración la difusión de tramas. Los mecanismos propuestos se han integrado en el switch híbrido AOSS debido a su sencillez, permitiéndose, con ello la exploración de nuevas vías para establecer nuevas funciones colaborativas en redes SDN.

7.1.3 Servicio de descubrimiento de caminos y topologías sobre redes con dispositivos híbridos SDN

Por último, se fijó como tercer objetivo el estudio, desarrollo e implementación de un servicio para el descubrimiento de topologías de red basado también en mecanismos de exploración de caminos por difusión de tramas. Este servicio debería ser compatible con redes de arquitectura SDN donde el conocimiento de la topología por parte del *Plano de Control* es de vital importancia para su correcto funcionamiento porque las decisiones se toman en base a la información topológica disponible.

En el capítulo 6 se ha presentado un nuevo protocolo de exploración de topologías para redes con arquitectura SDN y basadas en controlador. Este protocolo se presenta como una alternativa al protocolo OFDP derivado del estándar LLDP y, del cual se reutilizan reutilizado el formato de sus tramas de descubrimiento. Del mismo modo, también se han presentado dos implementaciones distintas de este nuevo protocolo, en primer lugar, una implementación basada únicamente en controlador, especialmente diseñada para la arquitectura SDN; y en segundo lugar, una implementación especialmente diseñada para su uso con dispositivos híbridos SDN siguiendo la filosofía de mecanismos colaborativos entre la *Capa de Control* y la *Capa de Infraestructura* inicialmente propuesta en el capítulo 4.

Con el objetivo de comprobar el correcto funcionamiento del protocolo propuesto, así como estudiar su rendimiento, se ha sometido al mismo a una serie de pruebas. Cuyos resultados demuestran que el protocolo es capaz de facilitar la información necesaria para que el controlador SDN pueda formar el grafo de la topología de la red de forma correcta. También han mostrado como la sobrecarga introducida (medida en número de paquetes de control) por el nuevo protocolo para obtener dicha información es menor que en OFDP (protocolo de referencia). Además, el nuevo protocolo es capaz de facilitar la información al controlador para la creación de un árbol de confluencia con raíz el nodo desde el que se lance el protocolo. Este árbol puede ser usado por el controlador para encaminar el tráfico, sin que para ello necesite realizar un procesamiento extra.

Tras el análisis de los resultados, se está en disposición dar por conseguido el objetivo sobre el estudio y la implementación de un nuevo protocolo capaz de proporcionar la información sobre la topología de la red al controlador, junto a cierta capacidad de encaminamiento sin coste adicional.

7.2 Trabajos futuros

La línea de trabajo seguida en esta tesis abre el camino a diferentes vías de investigación que se pretenden abordar en el futuro cercano.

Respecto al bloque de exploración de caminos, donde se han presentado TCP-Path y TFE, se pretende continuar con la línea de descubrimiento de caminos aplicando mecanismos de exploración y bloqueo en la capa de red en lugar de en la capa de enlace. Asimismo, también se piensa continuar con el estudio de exploración de caminos en redes híbridas aplicando estos conocimientos a los nuevos paradigmas de programación de conmutadores avanzados, como, por ejemplo, P4, ya sea a través del modelo emulado Behavioral Model version 2 (BMv2) o de la implantación de los diferentes sistemas en un entorno *hardware* como *NetFPGAs*.

Respecto al bloque de descubrimiento de topologías gracias al cual se ha presentado el protocolo TEDP se propone continuar aplicando el mecanismo de descubrimiento mediante difusión pero esta vez a redes híbridas compuestas por dispositivos SDN y no-SDN, estos últimos sin la capacidad de comunicarse directamente con el *Plano de Control*. Este nuevo mecanismo puede ser una interesante manera de descubrir topologías con características especiales tales como la movilidad de dispositivos , o la existencia de enlaces unidireccionales y bidireccionales en redes inalámbricas. Esto último provoca que los grafos de la red sean direccionales y, por tanto, la información topológica de la dirección de los enlaces se convierte en un aspecto fundamental para su correcta gestión.

Capítulo 8

Conclusions and future work

In this chapter, it is presented the conclusions obtained from this thesis, moreover, some future lines derived from the research carried out here are detailed as well.

8.1 Conclusions

As it has been seen in this thesis, data networks are in continuous evolution and, in general, this fact provokes an increase in its complexity. This thesis aims to alleviate some of these problems that the networks suffer despite the use of new network architectures. These problems could be provoked by the rise in the number of users and/or devices and/or services that can make use of the network.

This thesis starts by proposing a hybrid recovery mechanism to increase the network resilience (chapter 4). This mechanism uses the knowledge that the SDN *Control Plane* has about the network to efficiently deploy a distributed recovery mechanism. This approach minimizes the impact in the SDN *Control Layer*. The results show the benefits of the new mechanism in terms of recovery time and SDN *Control Plane* processing in comparison to an SDN centralised mechanism and a legacy distributed mechanism.

Later, this thesis has presented the TCP-Path protocol and, its variant, TFE (chapter 5) with the objective of improving the performance of legacy switches, and its integration in hybrid switches due to its simplicity. Both proposals can establish paths for TCP flows in an efficient way. The results obtained show that the performance in terms of BW and FCT improves the obtained performance with respect to ECMP. TFE achieves higher scalability than TCP-Path by applying the TCP-Path mechanism only to a small subset of TCP flows (those that carry very large amounts of traffic, called *Elephant* flows), which also slightly reduces its performance.

Finally, this thesis, by reusing the network exploration paradigm has proposed and developed a new protocol for SDN networks, named TEDP (chapter 6). TEDP is able to provide the topological information and a distribution tree to forward frames in a

discovered network. This proposal reduces the cost associated with the exploration process and improves the collected information after a discovery process started by the SDN *Control Layer* with respect to OFDP, which is the 'de facto' standard.

Even being the same objective for all contributions, the application of the network exploration paradigm to the different protocols and mechanisms proposed in this thesis, the results derived from the work embraces different features of the communication networks. For this reason, this chapter is divided into three different sections.

8.1.1 Recovery mechanism in hybrid SDN/Distributed environments

The first objective in this thesis was to study, develop, and implement a new mechanism for both SDN/Legacy network architectures to carry out a recovery when network failures appear. In chapter 4, the proposed recovery mechanism takes advantage of the features from legacy and SDN architectures to enhance previous recovery mechanisms. This mechanism uses the knowledge that the SDN *Control Layer* has about the network to locate the destination of a flow affected by a failure, and the legacy network ability to select new paths between a given pair of nodes using the network exploration paradigm based on the flooding of frames.

The new hybrid recovery mechanism has been compared with a fully distributed recovery mechanism, in which the paths affected by any failure are recovered using various exploration mechanisms and a centralized mechanism, in this case, a traditional SDN recovery based on computational calculation over the retrieved topological information.

The results obtained in tests show that the new hybrid mechanism works as expected since it is able to find a new path between a source and a destination when the network has some failures. The recovery time is under 50ms which as considered as a good value [87] when the network load is the same or less than 70 %, instead of if the load is higher than this value, we need some QoS mechanism to alleviate the problem. Moreover, these results also show that the new hybrid mechanism decreases the time that the *Control Layer* needs to carry out the recovery. Additionally, the recovery time of the new hybrid mechanism always outperforms the centralised approach as well as the distributed recovery under medium and high network loads.

With this work, the viability of hybrid solutions has been demonstrated and how its use can decrease the SDN *Control Layer* load. Hence, this thesis considers that the objective of the study and implementation of a new recovery mechanism for hybrid networks (SDN/Legacy) has been achieved.

8.1.2 Path discovery protocols TCP-Path and TFE

As the second objective, this thesis decided to develop and implement a new protocol capable of selecting low-latency paths for TCP flows between two end devices of the

network. Moreover, this mechanism must use an exploration process based on flooding to create paths. And also, it must be specially designed for high-performance network architectures, such as the data center networks.

In chapter 5, TCP-Path and its variant TFE are proposed to discover paths for TCP flows. TCP-Path discovers and establishes a new path for each new TCP flow that travels across the network. Whereas, TFE establishes new paths only for a subset of all TCP flows that cross the network. Both protocols select the lower-latency path that exists between two end-devices using exploration and blocking mechanisms [44]. In this chapter, a recovery mechanism is also proposed for both protocols. The importance of this type of mechanism is well-known by the community to guarantee the resilience of the network under eventual failures. The mechanism presented is an easy, fast and efficient process that takes advantage of the network exploration to generate a new bidirectional path between a source and a destination of TCP flows broken by failures based on latency.

Both protocols were tested to check their performance in terms of BW and FCT. The results show how both protocols (TCP-Path and TFE) are able to establish different paths in the right way. This fact fulfills the basic objective. The performance of both protocols in terms of BW and FCT is higher than the ECMP performance. This fact is because TCP-Path discovers each path in the exact moment that the TCP flows appear in the network. The obtained load balance in the network is better since TCP-Path considers the current status of the network to select the path. Likewise, the results show that the TFE performance is slightly lower than TCP-Path but way way higher than the ECMP performance since the TCP-Path mechanism is only applied to *Elephant* flows.

On the other hand, although TCP-Path improves ECMP performance, it has a scalability problem because it needs to insert a new rule into the switch forwarding table per each new TCP flow. Hence, if the network load increases, the state of each node increases as well. Moreover, the results also show that the size of the network is directly related to the number of flooding packets that the protocol needs. Likewise, the results show how the performance of TFE is somewhat lower than the TCP-Path protocol but its scalability is higher because it only needs to explore a new path for a small subset of all flows, which reduces the state (number of entries into the forwarding tables) of each node and the number of broadcast packets as well.

As it has been observed, the obtained results provide enough reasons for this thesis to consider achieved the second objective regarding the study and implementation of a new path discovery protocol based on network exploration. Moreover, the proposed mechanisms have been integrated into the AOSS hybrid switch for its later use in collaborative functions in SDN networks.

8.1.3 Path and topology discovery service for hybrid networks

Finally, this thesis sets as the third and last objective the study, development, and implementation of a new mechanism capable of discovering the network topology using different processes based on network exploration by flooding. This mechanism has been designed specially for the SDN networks where the knowledge of the network topology is vital to guaranty the correct work of the network.

In chapter 6, TEDP, a new topology exploration protocol for SDN networks, is presented. This protocol proposes an alternative to OFDP which is based on the standard LLDP and it uses its discovery frames. Likewise, two different implementations of this protocol are proposed, the first one is an implementation based on controller and specially designed for SDN architectures; and the second one is an implementation specially designed for hybrid SDN devices and networks.

The results obtained from the evaluation of TEDP show that the protocol is able to provide all the necessary information to the SDN controller so that it can generate the topology network graph with a lower number of packets in comparison to OFDP. Moreover, TEDP is also capable to provide a confluence tree rooted at any network node. This tree can be used by the controller to forward traffic and establish paths without requiring extra processing.

After analyzing the results, this thesis is ready to consider fulfilled the objective of the study and implementation of a new protocol capable of granting topological and forwarding information to the network controller.

8.2 Future work

The work line followed in this thesis opens different ways for future research which will be addressed in the near future.

Regarding path exploration protcols where TCP-Path and TFE have been presented, it is intended to continue with the path research line, applying exploration mechanisms in the network layer instead of in the link layer. Likewise, it is also planned to continue with the study of path exploration mecanisms in hybrid networks by applying this knowledge to new paradigms of switch programmanility, such as P4, either through the emulated model Behavioral Model version 2 (BMv2) or via the implementation in a specific hardware platform like *NetFPGAs*.

Regarding the topology discovery research line, this thesis proposes to continue applying the discovery mechanism by network exploration into hybrid SDN networks, composed by SDN and non-SDN devices, these last devices can not establish direct communication with the controller. This new mechanism could be a new way of discovering topologies with special characteristics such as device mobility in *wireless sensor networks*,

or the existence of unidirectional links, causing the network graph to be directional. Hence, the information about the direction of the links becomes a more important aspect to manage the network efficiently.

Bibliografía

[1] K. Pahlavan and P. Krishnamurthy, "IEEE Standard for Ethernet," *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pp. 1–5600, 05 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8457469

[2] T. y Pedraza, "IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges," *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pp. 1–281, 2004.

[3] D. Thaler and C. E. Hopps, "Multipath issues in unicast and multicast next-hop selection." *RFC*, vol. 2991, pp. 1–9, November 2000. [Online]. Available: http://dblp.uni-trier.de/db/journals/rfc/rfc2900-2999.html

[4] J. Moy, "OSPF Version 2," RFC 2328, Internet Engineering Task Force, United States, April 1998. [Online]. Available: https://doi.org/10.17487/RFC2328

[5] B. Nunes, M. Mendonca, Xuan-Nam Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] P. Berde *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014.

[7] J. Medved *et al.*, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, June 2014, pp. 1–6.

[8] "Ryu component-based Software Defined Networking framework," Accessed: 2020-07-30. [Online]. Available: https://github.com/osrg/ryu

[9] "OpenFlow - Open Networking Foundation," Accessed: 2020-07-30. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[10] GENI, "OpenFlow Discovery Protocol (OFDP)," 2010, Accessed: 2020-07-30. [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

[11] J. Alvarez-Horcajo, I. Martinez-Yelmo, E. Rojas, J. A. Carral, and D. Lopez-Pajares, "New cooperative mechanisms for software defined networks based on

hybrid switches," *Transactions on Emerging Telecommunications Technologies*, pp. e3150–n/a, 2017, e3150 ett.3150 bibtex: ETT:ETT3150. [Online]. Available: http://dx.doi.org/10.1002/ett.3150

[12] J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, and I. Martinez-Yelmo, "TCP-Path: Improving load balance by network exploration," in *Cloud Networking (CloudNet), 2017 IEEE 6th International Conference on*. IEEE, 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8071533

[13] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "TEDP: An Enhanced Topology Discovery Service for Software-Defined Networking," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1540–1543, Aug 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8376026

[14] J. Alvarez-Horcajo, D. Lopez-Pajares, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "Improving multipath routing of TCP flows by network exploration," *IEEE Access*, pp. 1–1, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8618377

[15] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to BOFUSS: The basic OpenFlow userspace software switch," *Journal of Network and Computer Applications*, vol. 165, p. 102685, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804520301594

[16] G. Ibáñez, J. Alvarez Horcajo, J. M. Arco, and J. M. Giménez, "Procedimiento cooperativo, entre puentes y controlador, de reparación de caminos en fallo y puente de red," Spanish, patent ES2 647 665 B2, PCT/ES2017/070 443, 2018. [Online]. Available: https://patentscope.wipo.int/search/es/detail.jsf?docId=WO2017220834

[17] G. F. Coulouris, *Sistemas distribuidos conceptos y diseño*, 3rd ed. Madrid, Spain: Adison-Wesley, 2001.

[18] D. De Ugarte, "El poder de las redes," El Cobre Ediciones, Tech. Rep., 2007, Accessed: 2020-07-30. [Online]. Available: http://www.pensamientocritico.org/davuga0313.pdf

[19] A. S. Tanembaum and M. Van Steen, "Sistemas Distribuídos: princípios e paradigmas," *2da. Edición*, 2007.

[20] Andrew S. Tanenbaum and David J. Wetherall, *Redes de Computadoras, 5ta Edición*, 5th ed. Madrid, Spain: PEARSON, 2015.

[21] P. García Teodoro, *Transmisión de datos y redes de computadores*, 2nd ed., ser. Always learning. Madrid, Spain: Pearson, 2014.

[22] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, April 1980.

[23] Jose Antonio Castillo, "Modelo OSI: que es y para que se utiliza," 2018, Accessed: 2020-07-30. [Online]. Available: https://www.profesionalreview.com/2018/11/22/modelo-osi/#Capa_2_Enlace_de_datos

[24] C. J. Kale and T. J. Socolofsky, "TCP/IP tutorial," RFC 1180, Internet Engineering Task Force, Jan. 1991. [Online]. Available: https://rfc-editor.org/rfc/rfc1180.txt

[25] J. Postel, "User Datagram Protocol," RFC 768 (Standard), Internet Engineering Task Force, Aug. 1980. [Online]. Available: https://rfc-editor.org/rfc/rfc768.txt

[26] ——, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[27] J. F. Kurose and K. W. Ross, *Redes de computadoras un enfoque descendente*, 7th ed. Madrid, Spain: Pearson Addison Wesley, 2017.

[28] M. Kazmi, A. Shamim, N. Wahab, and F. Anwar, "Comparison of TCP tahoe, reno, new reno, sack and vegas in IP and MPLS networks under constant bit rate traffic," in *International Conference on Advanced Computational Technology and Creative Media (ICACTCM)*, 2014, pp. 33–39.

[29] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[30] A. D. Vendictis, A. Baiocchi, and M. Bonacci, "Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP Reno network scenario," *Performance Evaluation*, vol. 53, no. 3, pp. 225 – 253, 2003, networking 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166531603000646

[31] Lisong Xu, K. Harfoush, and Injong Rhee, "Binary Increase Congestion Control (BIC) for fast long-distance networks," in *IEEE INFOCOM 2004*, vol. 4. IEEE, 2004, pp. 2514–2524. [Online]. Available: http://ieeexplore.ieee.org/document/1354672/

[32] D. Rattalino and A. Crespo, "Simulacion de Algoritmos de Control de Congestion en TCP bajo User Mode Linux," *Revista Tecnologia y Ciencia*, no. 27, pp. 214–223, 2015.

[33] F. J. I. García, "Estudio de las tecnologías SDN y NFV," Proyecto Final de Grado, Universitat Oberta de Catalunya, 2016.

[34] L. Arizmendi, "Ideas básicas sobre Software Defined Network (SDN)," Nov. 2013. [Online]. Available: http://luisarizmendi.blogspot.com.es/2013/11/software-defined-network-sdn.html

[35] O. N. Foundation, *OpenFlow Switch Specification 1.3.2.* Open Networking Foundation, Apr. 2013, Accessed: 2020-07-30. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf

[36] C. Cascone, "P4 support in ONOS," in *Proc. ONOS Build*, 2017, pp. 1–29, Accessed: 2020-07-30. [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=16122675

[37] "Funcionamiento de OpenFlow," Feb. 2015, Accessed: 2020-07-30. [Online]. Available: http://electivaredesavanzadas.blogspot.com.es/

[38] N. Figuerola, "SDN: Redes definidas por Software," Oct. 2013, Accessed: 2020-07-30. [Online]. Available: https://articulosit.files.wordpress.com/2013/10/sdn.pdf

[39] "Introducción a los grupos de control (cgroups) de Linux," Accessed: 2020-07-30. [Online]. Available: https://elpuig.xeill.net/Members/vcarceler/articulos/introduccion-a-los-grupos-de-control-cgroups-de-linux

[40] "ofsoftswitch13 - OpenFlow 1.3 switch," Accessed: 2020-07-30. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[41] "Open vSwitch," Accessed: 2020-07-30. [Online]. Available: http://openvswitch.org/

[42] "BEBA: BEhavioral Based Forwarding," Accessed: 2020-07-30. [Online]. Available: http://http://www.beba-project.eu/

[43] J. Álvarez-Horcajo, "Desarrollo de un switch híbrido OpenFlow/All-Path," Master's thesis, Universidad de Alcalá, Alcalá de Henares, Madrid, Spain, 2017. [Online]. Available: https://ebuah.uah.es/dspace/handle/10017/29680

[44] E. Rojas, G. Ibañez, J. M. Gimenez-Guzman, J. A. Carral, A. García-Martinez, I. Martinez-Yelmo, and J. M. Arco, "All-Path bridging: Path exploration protocols for data center and campus networks," *Computer Networks*, vol. 79, no. 0, pp. 120 – 132, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615000055

[45] R. S. Montero, E. Rojas, A. A. Carrillo, and I. M. Llorente, "Extending the Cloud to the Network Edge," *Computer*, vol. 50, no. 4, pp. 91–95, 2017.

[46] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.

[47] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch." in *NSDI*, 2015, pp. 117–130.

[48] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Cisco systems netflow services export version 9," in *Cisco Systems*. RFC 3954, October, 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3954

[49] M. Wang, B. Li, and Z. Li, "sFlow: Towards resource-efficient and agile service federation in service overlay networks," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 2004, pp. 628–635.

[50] B. Claise, M. Fullmer, P. Calato, and R. Penno, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101, Internet Engineering Task Force, 2005. [Online]. Available: https://tools.ietf.org/html/rfc5101

[51] P. Y. Fernádez, "Programación de redes SDN mediante el controlador POX," Ph.D. dissertation, Universidad Politécnica de Cartagena, CARTAGENA, Oct. 2015. [Online]. Available: http://repositorio.upct.es/bitstream/handle/10317/5254/tfg729.pdf?sequence=1

[52] D. Erickson, "The Beacon OpenFlow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[53] O. Akizuki and T. Fujikawa, "A Study of Emulator with Trema for Ad-hoc Networks," in *2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems*, 2015, pp. 225–229.

[54] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 254–265. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018466

[55] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855730

[56] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable Ethernet for Data Centers," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA:

ACM, 2012, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413183

[57] W. Stallings, "Software-Defined Networks and OpenFlow - The Internet Protocol Journal, Volume 16, No. 1." [Online]. Available: http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html

[58] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342446

[59] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: programming platform-independent stateful openflow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014.

[60] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp.–, Aug. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043164.1851224

[61] IEEE, "IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery," *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, pp. 1–146, 2016. [Online]. Available: https://standards.ieee.org/standard/802_1AB-2016.html

[62] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for OpenFlow," in *Communications (ICC), 2012 IEEE International Conference on*, Jun. 2012, pp. 6606–6610.

[63] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[64] "Networks Evolution: from Legacy to Virtual," Accessed: 2019-09-17. [Online]. Available: http://www.atrinet.com/networks-evolution-from-legacy-to-virtual/

[65] M. Rouse, "Hybrid SDN," Accessed: 2020-07-30. [Online]. Available: http://searchsdn.techtarget.com/definition/hybrid-SDN

[66] M. Huang and W. Liang, "Incremental SDN-Enabled Switch Deployment for Hybrid Software-Defined Networks," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, July 2017, pp. 1–6.

[67] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "Hybnet: Network manager for a hybrid network infrastructure," in *Proceedings of the Industrial*

Track of the 13th ACM/IFIP/USENIX International Middleware Conference, ser. Middleware Industry '13. New York, NY, USA: ACM, 2013, pp. 6:1–6:6. [Online]. Available: http://doi.acm.org/10.1145/2541596.2541602

[68] F. Farias, I. Carvalho, E. Cerqueira, A. J. Abelém, C. Esteve Rothenberg, and M. Stanton, "LegacyFlow: Bringing OpenFlow to Legacy Network Environments," Berlin, Germany., 2011. [Online]. Available: http://changeofelia.info.ucl.ac.be/pmwiki/uploads/SummerSchool/Program/poster_004.pdf

[69] R. Katiyar, P. Pawar, A. Gupta, and K. Kataoka, "Auto-Configuration of SDN Switches in SDN/Non-SDN Hybrid Network," in *Proceedings of the Asian Internet Engineering Conference*, ser. AINTEC '15. New York, NY, USA: ACM, 2015, pp. 48–53. [Online]. Available: http://doi.acm.org/10.1145/2837030.2837037

[70] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 333–345.

[71] H. work group, "Hybrid Switch Use Cases," Accessed: 2020-07-30. [Online]. Available: https://www.opennetworking.org

[72] I. Cisco Systems, "Cisco IOS Software Configuration Guide," 2001-2009, Accessed: 2020-07-30. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SXF/native/configuration/guide/swcg/cdp.pdf

[73] F. F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined etworks," in *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*, Dec 2014, pp. 1–8.

[74] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "softdp: Secure and efficient openflow topology discovery protocol," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–7.

[75] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, *A Distributed Algorithm for Topology Discovery in Software-Defined Networks*. Springer International Publishing, 2016, pp. 363–367.

[76] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Discovering the Network Topology: An Efficient Approach for SDN," *The Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 5, p. 101, 11 2016.

[77] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "ETDP: enhanced topology discovery protocol for software-defined networks," *IEEE access*, vol. 7, pp. 23 471–23 487, 2019.

[78] W. Wojdak, "Rapid Spanning Tree Protocol: A new solution from an old technology," *Reprinted from CompactPCI Systems*, 2003.

[79] A. F. De Sousa, "Improving load balance and resilience of ethernet carrier networks with IEEE 802.1s multiple Spanning Tree Protocol," in *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*. IEEE, 2006, pp. 95–95.

[80] D. Allan and N. Bragg, *802.1aq Shortest Path Bridging Design and Evolution: The Architect's Perspective*. Wiley, 2012. [Online]. Available: http://books.google.es/books?id=NRuOJ7xdcHMC

[81] R. C. Sofia, "A survey of advanced ethernet forwarding approaches," *IEEE Communications Surveys Tutorials*, vol. 11, no. 1, pp. 92–115, 2009.

[82] "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet," Accessed: 2020-07-30. [Online]. Available: http://mininet.org/

[83] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[84] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[85] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed Congestion-aware Load Balancing for Datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2740070.2626316

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

[87] N. Van Adrichem, B. Van Asten, and F. Kuipers, "Fast Recovery in Software-Defined Networks," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, Sep. 2014, pp. 61–66.

[88] "Assigned Internet Protocol Numbers," 01 2020, Accessed: 2020-07-30. [Online]. Available: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

[89] "ns-3 simulator," April 2017, Accessed: 2020-07-30. [Online]. Available: https://www.nsnam.org/

[90] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *Optical Interconnects Conference, 2013 IEEE*. Citeseer, 2013, pp. 49–50.

[91] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity Data Center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[92] "Getting started ryu 3.27 documentation," Accessed: 2020-07-30. [Online]. Available: http://ryu.readthedocs.org/en/latest/getting_started.html

[93] GEANTS, "GEANT topology map august 2017," GEANTS, Europe, Tech. Rep., 2017. [Online]. Available: https://www.geant.org/Resources/PublishingImages/GEANT_topology_map_august2017.pdf#search=August2017

[94] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "IEEE Data Port - Dataset of TEDP (experiments and results)," University of Alcala, Tech. Rep., 2018. [Online]. Available: http://dx.doi.org/10.21227/H2XT00

Apéndice A

Artículos y patente

Los artículos que abalan esta tesis son los siguientes:

1. Álvarez-Horcajo, Joaquin; Martínez-Yelmo, Isaías; Rojas, Elisa, Carral; Juan Antonio y López-Pajares, Diego (2017). New cooperative mechanisms for software defined networks based on hybrid switches. Transactions on Emerging Telecommunications Technologies, 28(8), e3150. (JCR Q3) [11]

2. Álvarez-Horcajo, Joaquín; López-Pajares, Diego; Arco, José Manuel ; Carral, Juan Antonio y Martínez-Yelmo, Isaías (2017). TCP-Path: Improving load balance by network exploration. IEEE 6th International Conference on Cloud Networking (CloudNet), 1-6. (conference with *Best Paper Award*) [12]

3. Rojas, Elisa; Álvarez-Horcajo, Joaquín; Martínez-Yelmo, Isaías; Carral, Juan Antonio y Arco, José Manuel (2018). TEDP: An enhanced topology discovery service for software-defined networking. IEEE Communications Letters, 22(8), 1540-1543. (JCR Q2) [13]

4. Álvarez-Horcajo, Joaquín; López-Pajares, Diego; Martínez-Yelmo, Isaías; Carral, Juan Antonio y Arco, José Manuel (2019). Improving multipath routing of TCP flows by network exploration. IEEE Access. (JCR Q1) [14]

5. E. L. Fernandes, E. Rojas, J. Álvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, The Road to BOFUSS: The basic OpenFlow userspace software switch, Journal of Network and Computer Applications, vol.165, p. 102685, 2020. (JCR Q1) [15]

La Patente que abala esta tesis es la siguiente:

1. Guillermo Ibáñez, Joaquín Álvarez, José Manuel Arco y José Manuel Giménez. "Procedimiento cooperativo, entre puentes y controlador, de reparación de caminos en fallo y puente de red". Patente de invención. Nº de patente: ES20160000530. Fecha de Concesión: 17/04/2018. [16]