CONTRIBUCIÓN AL DISEÑO DE REDES DEFINIDAS POR SOFTWARE

*DOCTORADO EN TECNOLOGÍAS DE LA
INFORMACIÓN Y LAS COMUNICACIONES*

Adrián Flores de la Cruz

Director/es:

Pilar Manzanares López

Juan Pedro Muñoz Gea

Cartagena 2019

Universidad Politécnica de Cartagena

Departamento de Tecnologías de la Información y las Comunicaciones

Tesis Doctoral

Contribución al Diseño de Redes Definidas por Software

Adrián Flores de la Cruz

2019

**CONFORMIDAD DE SOLICITUD DE AUTORIZACIÓN DE DEPÓSITO DE
TESIS DOCTORAL POR EL/LA DIRECTOR/A DE LA TESIS**

D^a. PILAR MANZANARES LÓPEZ y D. JUAN PEDRO MUÑOZ GEA Directores de la Tesis doctoral “CONTRIBUCIÓN AL DISEÑO DE REDES DEFINIDAS POR SOFTWARE”

INFORMA:

Que la referida Tesis Doctoral, ha sido realizada por D. ADRIAN FLORES DE LA CRUZ, dentro del Programa de Doctorado en TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES, dando mi conformidad para que sea presentada ante el Comité de Dirección de la Escuela Internacional de Doctorado para ser autorizado su depósito.

X Informe positivo sobre el plan de investigación y documento de actividades del doctorando/a emitido por el Director/ Tutor (**RAPI**).

La rama de conocimiento en la que esta tesis ha sido desarrollada es:

Ciencias
Ciencias Sociales y Jurídicas
X Ingeniería y Arquitectura

En Cartagena, a 17 de julio de 2019

LOS DIRECTORES DE LA TESIS

PILAR|
MANZANA
RES|LOPEZ

Firmado digitalmente
por PILAR|
MANZANARES|LOPEZ
Fecha: 2019.07.17
14:13:11 +02'00'

JUAN
PEDRO|
MUÑOZ|GEA

Firmado
digitalmente por
JUAN PEDRO|
MUÑOZ|GEA
Fecha: 2019.07.17
16:10:40 +02'00'

COMITÉ DE DIRECCIÓN ESCUELA INTERNACIONAL DE DOCTORADO

CONFORMIDAD DE DEPÓSITO DE TESIS DOCTORAL
POR LA COMISIÓN ACADÉMICA DEL PROGRAMA

D/D^a. Jorge Larrey Ruiz, Presidente/a de la Comisión Académica del Programa Tecnologías de la Información y las Comunicaciones.

INFORMA:

Que la Tesis Doctoral titulada, “CONTRIBUCIÓN AL DISEÑO DE REDES DEFINIDAS POR SOFTWARE”, ha sido realizada, dentro del mencionado Programa de Doctorado, por D/D^a. Adrián Flores de la Cruz, bajo la dirección y supervisión del Dr./ Dra. Pilar Manzanares López.

En reunión de la Comisión Académica, visto que en la misma se acreditan los indicios de calidad correspondientes y la autorización del Director/a de la misma, se acordó dar la conformidad, con la finalidad de que sea autorizado su depósito por el Comité de Dirección de la Escuela Internacional de Doctorado.

- χ Evaluación positiva del plan de investigación y documento de actividades por el Presidente de la Comisión Académica del programa (RAPI).

La Rama de conocimiento por la que esta tesis ha sido desarrollada es:

- ☐ Ciencias
- ☐ Ciencias Sociales y Jurídicas
- χ Ingeniería y Arquitectura

En Cartagena, a 17 de julio de 2019

EL PRESIDENTE DE LA COMISIÓN ACADÉMICA

**JORGE|
LARREY|
RUIZ**

Firmado digitalmente por JORGE|
LARREY|RUIZ
Nombre de reconocimiento (DN):
cn=JORGE|LARREY|RUIZ,
serialNumber= [REDACTED]
givenName=JORGE, sn=LARREY
RUIZ, ou=Ciudadanos, o=ACCV,
c=ES
Fecha: 2019.07.17 12:17:36 +02'00'

Fdo: Jorge Larrey Ruiz

COMITÉ DE DIRECCIÓN ESCUELA INTERNACIONAL DE DOCTORADO

Universidad Politécnica de Cartagena

Departamento de Tecnologías de la Información y las Comunicaciones

Tesis Doctoral

Contribución al Diseño de Redes Definidas por Software

Autor

Adrián Flores de la Cruz

Directores

Pilar Manzanares López

Juan Pedro Muñoz Gea

Cartagena, Octubre 2019

Para mi familia

Abstract

Software-Defined Networking (SDN) is a concept of network architecture that has emerged over the last few years. Contrary to traditional architectures where the control plane and the data plane coexist in network devices such as routers and switches, this concept basically consists on moving the control plane outside network devices and leaves only the data plane inside. The control plane is managed and administered by logically centralized elements called controllers. Network devices become simple packet forwarding devices that can be programmed through rules that are defined by the controller. In SDN, communication between the control plane and the data plane is mainly established by the OpenFlow protocol. In this thesis, efficient solutions have been proposed to three lines of research related to SDNs: networking monitoring and discovering, energy efficiency and development of a non-traditional routing protocol.

Network monitoring is becoming more important as more Internet Service Providers and enterprises networks deploy real time services. Having an update view of the network and to measure network performance with metrics like throughput, packet loss and delay is the main task of network monitoring. Obtaining accurate and meaningful network statistics helps find service degradation due to congestion and enables implementation of routing optimization solutions.

Achieving energy efficiency has recently become a key topic for networking research due to the ever-increasing power consumption and CO_2 emissions generated by large data networks. This problem is becoming even more concerning and challenging given the drastic traffic increase expected over the next few years. Putting into sleep mode unused elements is an effective and widely-accepted strategy to reduce the consumption of data networks. In this context, Software-Defined Networking (SDN) can be seen as an engaging solution to achieve the long-awaited energy efficiency in current communications systems, since they allow an adaptable programmability applicable to this problem.

The host-to-host/content/service communication instead of the host-to-host communication offered by traditional Internet Protocol (IP) routing solutions is being demanded in the last few years. Nowadays, getting this type of communication directly at network level is an increasing demand in the framework of new networking scenarios, such as Internet of Things and data center scenarios. Based on Key-Based Routing (KBR) solutions which,

together with Distributed Hash Tables, have offered a way of providing content-sharing solutions in overlay networks for years now, in this thesis we propose a key-based routing solution directly implemented at the network layer, taking advantage of the potential of SDN technology.

Resumen

En los últimos años ha surgido un nuevo concepto de arquitectura de red conocido como Software-Defined Networking (SDN). Contrariamente a las arquitecturas tradicionales, donde el plano de control y el plano de datos coexisten en dispositivos de red tales como routers y switches, este nuevo concepto consiste básicamente en mover el plano de control fuera de dichos dispositivos, dejando sólo el plano de datos. El plano de control quedará por tanto gestionado y administrado por elementos centralizados lógicamente, denominados controladores. Los dispositivos de red se convierten en simples dispositivos de reenvío de paquetes que se programan a través de reglas definidas por el controlador. En SDN, la comunicación entre el plano de control y el plano de datos (dispositivos de red) está establecida principalmente por el protocolo OpenFlow (OF). En esta tesis se han propuesto soluciones eficientes a tres líneas de investigación relativas a las redes SDN: la monitorización y descubrimiento de la red, la eficiencia energética y el desarrollo de un protocolo de encaminamiento no tradicional.

La monitorización de la red está adquiriendo importancia como consecuencia de que cada vez más Proveedores de Servicios de Internet (ISP) y redes empresariales despliegan servicios en tiempo real. Tener una visión actualizada de la red y medir su rendimiento con métricas como el *throughput*, la pérdida de paquetes y el retardo, se presenta como la principal tarea de la monitorización de la red. La obtención de estadísticas de red precisas y significativas ayuda a encontrar la degradación del servicio debida a la congestión y además permite implementar soluciones para la optimización del enrutamiento.

Lograr la eficiencia energética se ha convertido recientemente en un tema clave para la comunidad investigadora debido al aumento del consumo de energía y de las emisiones de CO_2 generadas por las grandes redes de datos. Si además contemplamos el drástico aumento de tráfico previsto para los próximos años, las emisiones serán todavía más alarmantes. La puesta en modo *sleep* de los elementos no utilizados es una estrategia eficaz y ampliamente aceptada para reducir el consumo de las redes de datos. En este contexto, las redes definidas por software (SDN) pueden considerarse como una solución atractiva para lograr la tan necesitada eficiencia energética en los sistemas de comunicaciones actuales, ya que permiten una programabilidad adaptable para este problema.

En los últimos años, la comunicación host-a-host/contenido/servicio está siendo más

demandada que la comunicación host-a-host ofrecida por las soluciones de enrutamiento tradicionales de Protocolo de Internet (IP). En la actualidad, conseguir este tipo de comunicación directamente a nivel de red tiene una demanda creciente en el marco de nuevos escenarios, como el *Internet of Things* y los escenarios de data-centers. Apoyándonos en soluciones Key-Based Routing (KBR) que, junto con las Tablas de Hash Distribuidas, han ofrecido una forma de proporcionar soluciones de compartición de contenido en redes overlay desde hace años, en esta tesis se propone una solución de enrutamiento basada en claves que se implementa directamente en la capa de red, aprovechando así el potencial de la tecnología SDN.

Agradecimientos

En primer lugar, agradecer al Ministerio de Economía y Competitividad haberme concedido la beca con la cual poder trabajar en mi doctorado durante tres años. Sin su apoyo financiero hubiera sido muy difícil poder haberme centrado en el desarrollo de este trabajo.

Gracias a mi mujer Fuensanta, que se ha volcado en todo momento en ayudarme, sufriendo junto a mí en las épocas más duras, siempre con sus palabras de ánimo y con su inigualable sonrisa en la cara. Sin toda la fuerza y el apoyo que me ha transmitido seguramente no hubiera sido capaz de acabar el doctorado.

Papá, Mamá y Laura, a pesar de estar separados durante mucho tiempo, siempre me habéis transmitido vuestra experiencia para afrontar los problemas, así como ofreciéndome increíbles consejos que me han servido tanto para la realización del doctorado como para tomar decisiones en mi día a día.

Por último, agradecer a todo el personal de la UPCT por ofrecerme en todo momento su ayuda, en especial a mis dos directores Pilar y Juan Pedro.

Índice general

Abstract	I
Resumen	III
Agradecimientos	V
Lista de Figuras	XI
Lista de Tablas	XV
Lista de Acrónimos	XVII
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura y contribuciones de esta tesis	3
2. Redes Definidas por Software	5
2.1. Introducción	5
2.2. Arquitectura SDN	5
2.3. OpenFlow	8
2.4. Mininet	13

3. Monitorización y Descubrimiento de la red mediante SDN	15
3.1. Introducción	15
3.2. Métodos de medición actuales	16
3.3. Mecanismos de monitorización de tráfico con soporte de fallos de red para redes SDN	17
3.3.1. Introducción	17
3.3.2. Propuesta de mejora de OpenNetMon	18
3.3.3. Evaluación de la solución propuesta	19
3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow	22
3.4.1. Introducción	22
3.4.2. Descubrimiento de la topología en redes OpenFlow SDN	23
3.4.3. Descubrimiento de hosts en redes OpenFlow	25
3.4.3.1. Consideraciones técnicas previas	25
3.4.3.2. Búsqueda proactiva de hosts	26
3.4.3.3. Seguimiento de las conexiones de los hosts	28
3.4.4. Evaluación del mecanismo de descubrimiento de hosts	29
3.5. Conclusiones	33
4. Influencia de las redes SDN en el ahorro energético	35
4.1. Introducción	35
4.2. Evolución del enrutamiento energético en las redes SDN	36
4.3. Optimización del consumo de energía en redes SDN	38
4.3.1. Introducción	38
4.3.2. Propuesta Energy-Aware	38
4.3.3. Algoritmos Heurísticos	42
4.4. Conclusiones	46

5. Encaminamiento no IP basado en claves en redes SDN OpenFlow	47
5.1. Introducción	47
5.2. Descripción de la solución de encaminamiento OFC-KBR	48
5.2.1. Introducción	48
5.3. Tecnologías existentes empleadas en OFC-KBR	49
5.3.1. Mecanismo de búsqueda Chord	49
5.3.2. Representación de rangos basada en expansión de prefijos	50
5.4. OFC-KBR	52
5.4.1. Asignación de identificadores virtuales	52
5.4.2. Información de encaminamiento basada en el uso de claves	53
5.4.3. Diferencia entre rutas físicas y rutas virtuales	56
5.5. Unión de terminales al sistema y definición del mecanismo de resolución de nombres	57
5.5.1. Definición de los paquetes JOIN	57
5.5.2. Mecanismo de resolución de nombres	58
5.5.3. Ejemplo de unión y de resolución de nombre	58
5.6. Evaluación del sistema propuesto	60
5.6.1. Conversión de las finger tables en rangos	61
5.6.2. Ocupacion de tablas OpenFlow	62
5.6.3. Rendimiento de la red	64
5.6.3.1. Optimización del enrutamiento	64
5.6.3.2. Throughput del enrutamiento	65
5.7. Propuesta de modificación del standard OpenFlow para mejorar el rendi- miento de OFC-KBR	67
5.8. Conclusiones	68
6. Conclusiones	69

Apéndice A	71
Bibliografía	75

Lista de Figuras

2.1. Arquitectura SDN	6
2.2. Comunicación entre un dispositivo de red y un controlador mediante el protocolo OpenFlow	8
2.3. Formato de entrada de una <i>flow table</i> definida en el estándar OpenFlow 1.3	9
2.4. Componentes principales de una entrada de la group table	11
2.5. Paquetes pasando por multiples tablas, en el proceso <i>pipeline</i>	12
3.1. Configuración de una red SDN basada en la tecnología OpenNetMon . . .	19
3.2. Detalle de la ruta principal entre host1 y host2	20
3.3. Retardo y <i>throughput</i> correspondiente a la transmisión de video entre host1 y host2 a través de la ruta inicial	21
3.4. Detalle de la ruta alternativa punteada	21
3.5. Retardo y <i>throughput</i> correspondientes a la transmisión de vídeo entre host1 y host2 en un escenario en el que switch 2 falla en el segundo 200 .	22
3.6. Proceso de descubrimiento de topología en redes SDN basadas en OpenFlow	24
3.7. Estructura de los campos <i>config</i> y <i>state</i>	25
3.8. Topología Fat-tree	30
3.9. Medición del ancho de banda de flujo obtenida en el último switch s5 . .	31

3.10. Velocidad de transmisión y recepción de los puertos de los switches s2, s5, s1 y s4	32
4.1. Consumo total de energía en una red de 70 nodos	43
4.2. Consumo total de energía en una red de 20 nodos	44
4.3. Porcentaje de energía ahorrada	45
4.4. Incremento de la potencia total cuando se activan nuevas demandas	45
5.1. Un ejemplo de búsqueda en un sistema Chord, donde $m = 6$ bits. El nodo 8 está buscando un objeto cuyo identificador clave es el 53	50
5.2. Ejemplo de topología física de una red SDN y la correspondiente asigna- ción de <i>vids</i> , donde los <i>vids</i> tienen una longitud de 10 bits	53
5.3. Teniendo en cuenta la topología física mostrada en la figura 5.2, esta fi- gura muestra la topología virtual obtenida con OFC-KBR. Las tablas de la parte inferior muestran un ejemplo de los rangos obtenidos correspon- dientes a los switches <i>dp2</i> y <i>dp6</i>	55
5.4. Implementación de rutas físicas utilizando tecnología MPLS	56
5.5. Cronograma que muestra la unión de <i>h1</i> y <i>h3</i> , y la resolución de nombre involucrada en el envío de mensajes de <i>h1</i> a <i>h3</i> y de <i>h3</i> a <i>h1</i>	59
5.6. Topologías. SwitchL3 (39 nodos y 62 enlaces), Dfn (56 nodos y 87 enla- ces) and GtsCe (148 nodos y 192 enlaces)	60
5.7. Distribución de probabilidad del número de rangos obtenidos para un no- do. $L = 32$ bits. (a) Para una simulación usando Mininet. (b) Usando el modelo matemático descrito en el Apéndice A	61
5.8. Topología Switch-L3 (39 nodos y 62 enlaces)	62
5.9. Topología DFN (56 nodos y 87 enlaces)	63
5.10. Topología GtsCe (148 nodos y 192 enlaces)	64

LISTA DE FIGURAS

5.11. Número de entradas en caché en el espacio del kernel. Como se ha co-
mentado en la sección 5.6.2, el aumento de las entradas de los switches
49 a 52 con respecto al resto de switches de la topología DFN se debe a la
posición y al alto grado de conectividad de estos switches. **(a)** Topología
SwitchL3. **(b)** Topología DFN 66

A.1. Distribución de probabilidad del número de rangos obtenidos por simulación 73

A.2. Distribución de probabilidad del número de rangos obtenidos utilizando
el modelo matemático 73

Lista de Tablas

3.1. Descubrimiento proactivo de host. Pseudocódigo	27
4.1. Algoritmo heurístico propuesto	42
4.2. CALCULATE POWER (path,demand)	43
5.1. Match fields de la especificación OpenFlow 1.3	51
5.2. Algoritmo para obtener los rangos representables mediante un valor y una máscara de bits utilizando la expansión de prefijos	52
5.3. <i>Finger Table</i> de un nodo cuyo identificador virtual es n	54
5.4. Rangos y acciones asociadas correspondientes a un nodo de identificador virtual n . $ft(n + 2^i)$ hace referencia al valor correspondiente a la entrada n $+ 2^i$, indicando el switch responsable del identificador $n + 2^i$	54
5.5. Routing Stretch	65

Lista de Acrónimos

API	Application Programming Interface
ARP	Address Resolution Protocol
DHT	Distributed Hash Table
DPI	Deep Packet Inspection
ELCP	Extended LCP
GreenTE	Green Traffic Engineering
ICMP	Internet Control Message Protocol
ICT	Information and Communication Technologies
ID	IDentifier
IDS	Intrusion Detection System
IEA	International Energy Agency
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ILP	Integer Linear Programming
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
IT	Information Technology
KBR	Key-Based Routing

LCP	Longest Common Prefix
LLDP	Link Layer Discovery Protocol
LPM	Longest Prefix Match
LSP	Label Switching Path
MAC	Media Access Control
MIB	Management Information Base
MILP	Mixed Integer Linear Programming
MPLS	Multi-Protocol Label Switching
OFC-KBR	OpenFlow Compatible Key-Based Routing
ONF	Open Network Foundation
OF	OpenFlow
OS	Operating System
OSPF	Open Shortest Path First
OVS	OpenvSwitch
PIDR	Point Intersection Disjoint Ranges
QoS	Quality of Service
RS	Routing Stretch
RTT	Round Trip Time
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SP	Shortest Path
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TE	Traffic Engineering
TLS	Transport Layer Security
UDP	User Datagram Protocol
VIRO	Virtual Identifier Routing
VLAN	Virtual LAN

Introducción

1.1. Motivación

En la actualidad, Internet tiene un papel protagonista en nuestras vidas y ha revolucionado la forma en la que aprendemos, nos entretenemos e incluso la forma en la que nos relacionamos. El principal responsable de ésto es el frenético avance tecnológico que venimos presenciando desde hace unos años, y que ha permitido, entre otras cosas, el acceso a Internet desde una amplia gama de dispositivos, cada vez más accesibles y móviles. Prueba de ello es que, en los últimos años, tanto el número de dispositivos conectados a Internet como el volumen de tráfico generado por los mismos ha aumentado de un modo más que considerable.

En su intento de poder atender el continuo y creciente ritmo de la demanda, los Proveedores de Servicios de Internet (ISP) a menudo se ven obligados a reconfigurar su propia red, bien actualizando los dispositivos ya existentes, o bien instalando y configurando nuevos dispositivos. Así mismo, cada vez es más frecuente y necesaria la transformación de las políticas de alto nivel en comandos de configuración de bajo nivel específicos de cada dispositivo, por parte de los distintos operadores. Obviamente, este proceso de adaptación de niveles en todos y cada uno de los dispositivos, presenta una gran complejidad y es propenso a errores debido a la gran variedad de dispositivos existentes en cualquier red. En definitiva, la reconfiguración automática no existe en las redes actuales, lo que sin duda plantea un problema a los ISPs a la hora de poder atender y adaptarse a la creciente demanda de los usuarios de Internet.

A fin de acelerar el proceso de innovación y adaptación, los operadores necesitan más flexibilidad para poder controlar y personalizar en la medida de lo posible cualquiera de los dispositivos existentes en la red. El paradigma SDN (Software-Defined Networking) [1] tiene su origen en diversos trabajos del mundo de la investigación que demostraron la necesidad de construir redes programables como una forma práctica de experimentar con nuevos protocolos en redes existentes. Desde su aparición, las redes SDN se han ganado la atención y consideración tanto del mundo académico como de la industria. El concepto de SDN está respaldado por grandes marcas del mundo de Internet como: Google, Cisco,

HP, Juniper o NEC, y también por organizaciones de estandarización como Open Networking Foundation (ONF) [2] e Internet Engineering Task Force (IETF), por lo que se puede afirmar que este modelo de red tiene y tendrá una influencia cada vez mayor en este ámbito. Sin duda alguna, la propuesta o aparición del protocolo OpenFlow [3], en 2008, fue su empuje definitivo. OpenFlow permite llevar a cabo la división entre dos planos distintos, uno para los datos y otro de control, con el fin innovador de desacoplar, lógicamente y físicamente, las distintas tareas y funciones propias de las redes.

SDN nos ofrece los beneficios de disponer de un enfoque centralizado para la gestión de la red. A diferencia de las redes tradicionales, el administrador de una red SDN modificará las políticas de la red desde el controlador. Éste se encargará de la traducción de las nuevas políticas en reglas y de que éstas sean instaladas en los distintos dispositivos de la red. Además, las redes basadas en SDN ofrecen un nivel de flexibilidad nunca antes visto, permitiendo realizar una gestión de nivel de flujo extremadamente precisa *fine-grained*, ideal para servicios con estrictos requisitos de calidad de servicio (QoS). Otra característica de SDN es su gran capacidad de adaptación a entornos que requieran atender a grandes volúmenes de tráfico cambiante o aleatorio, a fin de lograr un uso eficiente de los recursos que disponga la red. En SDN, los dispositivos de red serán de uso genérico, es decir, no estarán diseñados para una función de red específica como puedan ser los firewalls o los routers. Aprovechando la visión global que dispone el controlador, se podrá reconfigurar, vía software, la topología de red e incluso se podrá modificar el rol de los diferentes dispositivos en tiempo real. Como ya hemos comentado, en las redes tradicionales esta tarea es muy compleja, ya que los dispositivos de reenvío son inflexibles debido a la implementación de las reglas de enrutamiento. En cuanto a las funcionalidades de medición y monitorización, SDN permite obtener algunas estadísticas de tráfico al instante, para así poder optimizar el funcionamiento de la red, tareas que en algunos casos y en las grandes redes tradicionales, son inviables.

1.2. Objetivos

El principal objetivo de esta tesis doctoral es realizar un estudio en profundidad de las redes SDN que nos permita aportar propuestas de mejora, gracias a la potencialidad de dicha tecnología. Este objetivo está originado por el gran interés que las redes SDN han suscitado en la industria IT. Prácticamente todas las empresas fabricantes de equipos, además de muchas de las desarrolladoras de software e incluso empresas dedicadas a soluciones de virtualización, han desarrollado sus propias soluciones basadas en SDN.

A continuación vamos a reiterar y detallar las tres líneas de investigación que hemos decidido desarrollar:

- Monitorización de la red, midiendo parámetros tan esenciales como:
 - rendimiento de la red (*throughput*)
 - delay

1.3. Estructura y contribuciones de esta tesis

- descubrimiento de la posición de host en la topología de red
- Búsqueda de soluciones que nos permitan un ahorro del consumo de energía en redes SDN. Para ello hemos identificado diferentes objetivos a lograr:
 - enrutar de un modo eficiente, energéticamente hablando, las comunicaciones en redes SDN.
 - aumentar el ahorro de consumo de energía, cuantificando y considerando, en todo momento, el impacto que dicho ahorro energético pueda causar en la disminución del rendimiento de la red.
- Propuesta y diseño de un protocolo de encaminamiento no IP adecuado para redes SDN. Debido a la evolución de la demanda surgida en estos últimos años, cada vez se requieren más comunicaciones de host a host/contenido/servicio, en lugar del tipo de comunicación de host a host ofrecido por las soluciones de enrutamiento tradicionales.

1.3. Estructura y contribuciones de esta tesis

La estructura de la tesis está dividida en seis capítulos. El capítulo 1 presenta la motivación para este trabajo, así como las contribuciones realizadas en la tesis. En el capítulo 2 se describen los aspectos principales de redes SDN y del protocolo OpenFlow. El capítulo 3 presenta el trabajo realizado en el campo de la monitorización de redes bajo el paradigma SDN. En el capítulo 4 se introduce una nueva solución para el encaminamiento basado en eficiencia energética en redes SDN. El capítulo 5 describe un nuevo método de comunicación no IP basado en claves compatible con el protocolo OpenFlow. Por último en el capítulo 6 exponemos las conclusiones obtenidas en la tesis.

Las contribuciones realizadas a lo largo de la tesis son:

- A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez and J. Malgosa-Sanahuja. “Network failures support for traffic monitoring mechanisms in software-defined networks”. En: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. Abr. de 2016, págs. 691-694.
- P. Manzanares-Lopez, J. P. Muñoz-Gea, F. Delicado-Martinez, J. Malgosa-Sanahuja and A. F. de la Cruz. “Host Discovery Solution: An Enhancement of Topology Discovery in OpenFlow based SDN Networks”. En: 13th International Joint Conference on e-Business and Telecommunications Ene. de 2016, págs. 80-88.
- J. Malgosa-Sanahuja, A. F. de la Cruz, J. P. Muñoz-Gea and P. Manzanares-Lopez. “Solving IP exhaustion with maskless inter-domain routing (MIDR) scheme”. En: 2017 Sixth International Conference on Future Generation Communication Technologies (FGCT). Ago. de 2017, págs. 1-5.

- A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez and J. Malgosa-Sanahuja. “Optimization of Power Consumption in SDN Networks”. En: 2017 The Ninth International Conference on Emerging Networks and Systems Intelligence (EMERGING 2017). Nov. de 2017, págs. 12-15.
- A. F. de la Cruz, P. Manzanares-Lopez, J. P. Muñoz-Gea and J. Malgosa-Sanahuja. “OpenFlow Compatible Key-Based Routing Protocol: Adapting SDN Networks to Content/Service-Centric Paradigm”. En: Journal of Network and Systems Management, vol. 27, num. 3, Jul. de 2019, págs. 730-755.

Redes Definidas por Software

2.1. Introducción

La aparición de Software Defined Networking (SDN) [1] o Redes Definidas por Software ha supuesto uno de los cambios de paradigma más significativos que la industria de las redes ha visto en los últimos años. El concepto SDN ha sido impulsado por la necesidad de mantenerse al día con los nuevos requisitos de red de las tendencias emergentes de computación en nube virtualizada, movilidad y aplicaciones de big data. Los objetivos de SDN incluyen la capacidad de introducir innovaciones de red más rápido y también la posibilidad de simplificar y automatizar radicalmente la administración de redes grandes.

La Open Networking Foundation (ONF) [2] creada en el año 2011 y financiada por compañías como Deutsche Telekom, Facebook, Google, Microsoft, Verizon y Yahoo!, es una organización sin ánimo de lucro dedicada a la investigación e innovación de SDN a través del desarrollo de estándares abiertos. Según la ONF, Software Defined Networking (SDN) ofrece una arquitectura emergente y que además es dinámica, manejable, rentable y adaptable, es decir, ideal para la naturaleza de las aplicaciones actuales.

2.2. Arquitectura SDN

El concepto de redes SDN se fundamenta en una arquitectura alternativa a las redes tradicionales basada en la separación o desacoplo del plano de datos (encargado del envío de datos) y del plano de control (encargado de la gestión y configuración de los elementos de red). Para poder gestionar de forma óptima los diferentes dispositivos de red y adaptarlos a las exigencias y necesidades del comportamiento del tráfico requerido, se desarrollaron una serie de protocolos para así estandarizar una interfaz entre el elemento centralizado (o de control) y cada uno de dichos dispositivos de red.

Podemos destacar las siguientes características en la arquitectura SDN [4]:

- centralizada, al presentar el controlador una visión global de la red, lo que le otorga

una inteligencia centralizada de la misma,

- programable, ya que las funciones de reenvío (pertenecientes al plano de datos) están separadas de las funciones de control (pertenecientes al plano de control) permitiendo al controlador de la red SDN modificar directamente el comportamiento de los dispositivos de red,
- ágil, ya que, gracias a la existencia de un elemento de control centralizado los administradores pueden configurar y ajustar dinámicamente la configuración de los elementos de la red, pudiendo, de ese modo, hacer frente a las necesidades cambiantes según el tipo de tráfico demandado,
- configurable, mediante programación. SDN ofrece a los diferentes administradores de la red configurar, administrar, monitorizar y optimizar los recursos de red en tiempo real, mediante programas propios, dinámicos y automatizados,
- basada en estándares abiertos, lo que facilita la operatividad y el diseño de la red, al no tener que emplear múltiples protocolos específicos para cada uno de los dispositivos de red de los diferentes proveedores.

La figura 2.1 muestra la arquitectura SDN, que consta de tres planos o capas:

Figura 2.1: Arquitectura SDN

2.2. Arquitectura SDN

Plano de control

Será el responsable de la toma de decisiones acerca del modo en que los paquetes deberán ser reenviados por los distintos dispositivos de la red, y también de hacer llegar las instrucciones precisas a los dispositivos de red implicados y/o responsables para su correcta ejecución [5]. Estará formado por un conjunto, lógicamente centralizado, de controladores SDN que tendrán una visión global y abstracta (estructurada) de la totalidad de los elementos de la red, lo que permitirá al administrador de la misma poder aplicar políticas personalizadas que se traducirán en una configuración específica de los elementos que conforman la red. Para la comunicación entre el controlador y los dispositivos de red, la arquitectura SDN define un interfaz denominado *SouthBound Interface*.

Como ya hemos dicho, el controlador de la red SDN será la entidad inteligente que controlará todos y cada uno de los dispositivos de la red para poder implementar cualquiera de los servicios que pudieran ser demandados a dicha red [6]. Las funcionalidades del plano de control incluirán:

- el análisis previo (descubrimiento) y la posterior actualización y mantenimiento de la topología de red,
- la selección y asignación de las rutas para el envío de paquetes, pudiendo implementar además mecanismos de reconfiguración ante situaciones problemáticas,
- el recopilar la información actualizada de estado de la red.

Plano de datos

El plano de datos está compuesto por los elementos de red, tanto físicos como virtuales, que se encargan de las funciones propias de los requisitos del reenvío de paquetes. Los elementos del plano de datos son accesibles desde el plano de control mediante la *SouthBound Interface*. Esta interfaz permite que el controlador SDN y los dispositivos de red estén continuamente comunicados entre sí.

Plano de aplicación

En él residen tanto las aplicaciones como los servicios que definen el comportamiento de la red, y contendrá las aplicaciones de red que interactuarán conjuntamente con el controlador para lograr una función de red específica que satisfaga las necesidades del operador, tales como: la calidad de servicio, la seguridad, la virtualización y algunas otras funciones relacionadas con la ingeniería de tráfico.

Este plano permite especificar el comportamiento que el administrador o gestor de la red desea en función de los requisitos de las aplicaciones. Para ello, el plano de control proporciona al plano de aplicación una vista abstracta de la propia red a través de una *NorthBound Interface*.

2.3. OpenFlow

El protocolo OpenFlow (OF) [3], considerado como uno de los primeros estándares de las redes definidas por software, define un protocolo de comunicación entre el controlador SDN y los elementos de red (switches, routers, etc.), tanto físicos como virtuales, que componen la red.

Como ya hemos dicho anteriormente, en la arquitectura SDN, el controlador es el cerebro de la red. Haciendo uso de la *Application Programming Interface* o *API South-Bound* el controlador configurará la funcionalidad de los dispositivos de red. La figura 2.2 representa la conectividad entre un controlador y un dispositivo de red que implementa OpenFlow.

Figura 2.2: Comunicación entre un dispositivo de red y un controlador mediante el protocolo OpenFlow

Como se puede observar en la figura 2.2, la comunicación entre el controlador y el dispositivo de red OpenFlow se realizará a través de un canal seguro implementado sobre una conexión TCP. A través de dicho canal, el controlador y el elemento de red intercambiarán diferentes mensajes, definidos por el estándar, que permitirán al controlador establecer las reglas de funcionamiento adecuadas en los dispositivos. Además, como veremos más adelante, OpenFlow define una serie de tablas donde se almacenarán dichas reglas. En concreto, un dispositivo de red OpenFlow estará formado por una o varias *flow tables* y una *group table*, en las que se realizará una búsqueda cada vez que el dispositivo reciba un paquete de datos, para averiguar la acción a realizar con dicho paquete.

Flow tables

La *flow table* es la estructura de datos que reside en cada OpenFlow Switch esencial para realizar operaciones propias de coincidencia y encaminamiento de los paquetes que estén siendo tratados por dicho dispositivo de reenvío. Una *flow table* estará compuesta

2.3. OpenFlow

por un conjunto de entradas, cada una de las cuales tendrá la estructura descrita en la figura 2.3. Consideramos el formato de entrada definido por el estándar OpenFlow 1.3 puesto que es la versión con la que se ha desarrollado todo el trabajo de esta tesis.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags
--------------	----------	----------	--------------	----------	--------	-------

Figura 2.3: Formato de entrada de una *flow table* definida en el estándar OpenFlow 1.3

Cada entrada de la *flow table* estará compuesta por los elementos identificados en la figura anterior. El primero de ellos, *match fields*, contiene los valores que se emplearán en la comparación de las cabeceras del paquete recibido para decidir si la entrada es válida para dicho paquete.

El campo *prioridad* determinará la prioridad de ejecución de la entrada frente a otras en el caso de que un paquete recibido tenga más de una entrada válida.

El campo *counters* define una serie de contadores para realizar un seguimiento de los paquetes reenviados al aplicar dicha entrada.

Los *timeouts* definen el tiempo de vigencia de una entrada.

Finalmente, el campo *instructions* define qué hacer con el paquete entrante cuando se aplique dicha entrada.

El estándar OpenFlow 1.3 define los siguientes tipos de instrucciones:

- Apply-Actions action(s),
- Meter meter id,
- Clear-Actions,
- Write-Metadata metadata / mask,
- Write-Actions action(s),
- Goto-Table next-table-id,

De entre todas las instrucciones destacamos tres: las que permiten añadir (*write-actions*), eliminar (*clear-actions*) o ejecutar acciones inmediatamente *apply-actions*. También es importante definir la instrucción *goto-table*, que sirve para indicar la siguiente *flow table* que deberá procesarse antes de ejecutar el conjunto final de acciones que hay que aplicar al paquete recibido.

Volviendo a las acciones, desde que se recibe un paquete hasta que atraviesa todas las *flow tables* involucradas (efecto *pipeline*), el switch OpenFlow va acumulando, en lo que

se denomina *action set* las acciones que deberán finalmente aplicarse al paquete recibido.

Las acciones definidas por el estándar OpenFlow 1.3 son las siguientes:

- Output
- Set-Queue
- Drop
- Group
- Push-Tag/Pop-Tag

De todas estas acciones destacamos la acción *Output*, que define por qué puertos OpenFlow debe ser transmitido el paquete. Los puertos OpenFlow pueden ser puertos físicos, puertos lógicos o puertos reservados:

- físico, consiste en una interfaz de hardware de un switch (compatible con OpenFlow), donde suele conectarse, físicamente, la Ethernet,
- lógico, en realidad se trata de una interfaz virtual de un OpenFlow Switch, que no corresponde directamente con una interfaz hardware de un switch,
- reservado, que es una interfaz lógica que está configurada para un reenvío específico y puede estar asociado a uno o más puertos lógicos y físicos. No se necesita un switch completo para soportar cada uno de los puertos reservados, si no sólo aquellos que estén marcados como 'requeridos' y que especificamos a continuación:
 - ALL: representa a todos los puertos que el switch puede usar para reenviar un paquete específico,
 - CONTROLLER: representa el puerto utilizado para la comunicación con el controlador OpenFlow,
 - TABLE: representa el reenvío de paquetes a la siguiente *flow table* cuando ello sea necesario. Este tipo de puerto se utilizará durante el procesamiento *pipeline*,
 - IN_PORT: representa el puerto por donde llegará el paquete entrante,
 - ANY: valor especial utilizado en algunos comandos de OpenFlow cuando no se especifica ningún puerto (*wildcard*),
 - LOCAL: el puerto local permite que las entidades remotas interactúen con el switch y sus servicios de red a través de la red OpenFlow,
 - NORMAL: para procesar el paquete como un switch Ethernet tradicional,
 - FLOOD: para reenviar paquetes a través de todas las interfaces de conmutación, excepto a la interfaz de entrada.

2.3. OpenFlow

Group Table

Además de las *flow tables*, el estándar OpenFlow 1.3 define una *group table*, que facilita operaciones de paquetes más complejas y especializadas que no pueden realizarse fácilmente a través de una entrada básica de la *flow table*. La *group table* contiene entradas identificadas por un identificador de grupo, tal y como vemos en la figura 2.4. Una entrada de la *group table* se aplicará a un paquete recibido como consecuencia de una acción tipo *Group* definida por una entrada de una *flow table*.

Group Identifier	Group Type	Counters	Action Buckets
------------------	------------	----------	----------------

Figura 2.4: Componentes principales de una entrada de la group table

Puesto que durante la realización de esta tesis no ha sido necesaria su utilización, no detallaremos los campos y uso de esta tabla. Para una información detallada, consultar el estándar de OpenFlow 1.3.

OpenFlow Channel

Como definimos al comienzo de este apartado, el canal OpenFlow es la interfaz que conecta cada dispositivo OpenFlow con un controlador. A través de dicha interfaz, el controlador recibe mensajes de control y configura y gestiona el switch. El canal OpenFlow está usualmente encriptado usando TLS (*Transport Layer Security*), pero podría ser también ejecutado directamente sobre TCP.

El protocolo OpenFlow soporta tres tipos de mensajes:

- Controller-to-switch: estos mensajes se iniciarán en el controlador y serán usados para manejar o inspeccionar directamente el estado del switch. Los principales mensajes son:
 - Feature Request y Feature Reply
 - Configuration Request, Configuration Reply y Set Configuration
 - Table Mod, Flow Mod, Group Mod, Port Mod, Meter Mod
 - Multipart Request y Multipart Reply (empleados para obtener diferentes estadísticas)
 - Packet-Out
 - Barrier Request y Barrier Reply
 - Role Request y Role Reply
 - Set Asynchronous Configuration

- Asynchronous: estos mensajes serán inicializados por el switch y serán usados para actualizar los eventos de red, del controlador y los cambios de estado en el switch. Los posibles mensajes son:
 - Packet-In
 - Flow-Removed
 - Port-status
 - Error
- Symmetric: estos mensajes podrán ser inicializados tanto por el switch como por el controlador, y serán enviados sin solicitud alguna por parte de ninguno de ellos. Los posibles mensajes son:
 - Hello
 - Echo Request y Echo Reply
 - Experimenter

Procesado de un paquete en un switch OpenFlow

Como se mostró en la figura 2.2, un switch OpenFlow consta de una o mas *flow tables* y una *group table*. Cada *flow table* estará compuesta por diferentes entradas. Los paquetes entrantes buscarán las coincidencias en dichas tablas para así poder asegurar las acciones que hay que realizar. El protocolo OpenFlow permite establecer una secuencia lineal de *flow tables* que debe recorrer un paquete conocida como *pipeline*, tal y como vemos en la figura 2.5. El procesamiento *pipeline* definirá cómo interactúan los distintos paquetes con las diferentes *flow tables*.

Figura 2.5: Paquetes pasando por multiples tablas, en el proceso *pipeline*

Las *flow tables* de un switch OpenFlow están numeradas secuencialmente comenzando por el número cero (0). El proceso *pipeline* siempre comienza en la primera *flow table*, es decir, el paquete entrante se compara primero con las entradas de flujo de la *flow table* 0. Dependiendo del resultado obtenido en la primera *flow table*, se dirigirá a otras *flow tables*. Cualquier entrada de flujo sólo podrá dirigir un paquete a una *flow table* de numeración mayor, es decir, el procesamiento *pipeline* sólo podrá ir hacia adelante y no hacia

atrás. Al finalizar el procesamiento *pipeline* asociado a un determinado flujo, se ejecutarán el conjunto de acciones que se han ido añadiendo al *action set*.

2.4. Mininet

Mininet [7] [8] es una plataforma destinada a la emulación de redes ampliamente utilizada en la evaluación de propuestas orientadas a redes definidas por software. Dichas redes serán totalmente escalables y podrán llegar hasta cientos de nodos dependiendo de la configuración deseada. Dicha plataforma nos permitirá crear, interactuar, personalizar e incluso compartir, de una forma ágil y rápida, un determinado prototipo de red definido mediante software.

Mininet tiene la capacidad de emular distintos tipos de elementos de red (hosts, switches y enlaces) operando sobre un único núcleo de Linux, empleando para ello virtualización. Así mismo, Mininet es una herramienta plenamente compatible con el desarrollo y con la evolución tanto del protocolo OpenFlow como de los sistemas SDN en general.

A día de hoy, está plenamente activo y en continuo desarrollo, manteniendo un excelente soporte técnico y lógico. Además, es una herramienta de código abierto, con las innumerables ventajas que ello conlleva.

Monitorización y Descubrimiento de la red mediante SDN

3.1. Introducción

La tecnología SDN, de gran interés tanto para el ámbito de la investigación como para la industria, permite realizar una precisa y completa gestión de las redes. Para ello, en muchos casos, es necesario monitorizar diferentes parámetros que podrán ser empleados en la configuración y gestión de las redes con el objetivo de alcanzar requerimientos entre otros de calidad de servicio (QoS, *Quality of Service*). La monitorización de redes ha sido uno de los principales campos de estudio e investigación en estos últimos años debido a la complejidad de medir, en tiempo real y de una forma precisa, el tráfico generado en la red y debido además a la dificultad de desarrollar las estructuras adecuadas para realizar las mediciones.

Hoy en día, la monitorización en las redes OpenFlow, tanto del rendimiento (*throughput*) como del retardo (*delay*) asociado a los flujos, representa una importante y necesaria herramienta para la implementación de soluciones de ingeniería de tráfico (TE, *Traffic Engineering*). Los mecanismos de monitorización de flujo requerirán identificar, de una forma única, la ruta completa que siguen todos y cada uno de los paquetes de flujo a través de la red.

Por esta razón, los mecanismos de monitorización por flujo están generalmente vinculados al controlador, puesto que éste posee un conocimiento global de la topología de red así como de los flujos establecidos en la misma. Además, puesto que pueden ocurrir fallos en los dispositivos de la red, los mecanismos de monitorización deberán ser capaces de detectar y reaccionar ante estas situaciones, que implican modificaciones en las rutas.

Es por ello que el conocimiento de la red es esencial para realizar una monitorización completa del tráfico, proporcionando al plano de control una visión precisa del estado de la red. Gracias a esta información, el controlador nos permitirá llevar a cabo una gestión de red eficaz y precisa para diferentes propósitos (p.e. TE, seguridad).

En este capítulo presentamos dos contribuciones desarrolladas durante la evolución de esta tesis doctoral relacionadas con esta temática, que dieron lugar a las siguientes publicaciones:

- A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez and J. Malgosa-Sanahuja. “Network failures support for traffic monitoring mechanisms in software-defined networks”. En: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. Abr. de 2016, págs. 691-694.
- P. Manzanares-Lopez, J. P. Muñoz-Gea, F. Delicado-Martinez, J. Malgosa-Sanahuja and A. F. de la Cruz. “Host Discovery Solution: An Enhancement of Topology Discovery in OpenFlow based SDN Networks”. En: 13th International Joint Conference on e-Business and Telecommunications Ene. de 2016, págs. 80-88.

3.2. Métodos de medición actuales

Sin duda, en los últimos años, las técnicas de monitorización del tráfico ofrecido y cursado en las distintas redes han sido un foco de interés para la investigación. En la actualidad se emplean diferentes técnicas para medir parámetros tales como la utilización de los enlaces, el retardo extremo a extremo, el rendimiento de la ruta y la pérdida de paquetes enviados. Las dos tendencias principales de medición de estos parámetros son, por un lado, los sistemas de monitorización (tradicionales) que ya han sido desplegados y, por otro lado, los nuevos enfoques orientados a las redes SDN.

La primera línea de monitorización, es decir, la tradicional, describe los métodos usados por los ISPs para monitorizar sus infraestructuras. Estos métodos suelen estar bastante limitados, ya que no consiguen ofrecer una monitorización completa de la red, si no, solo una información parcial de ella.

Dentro de este grupo de monitorización podemos destacar varios métodos, como el *Simple Network Management Protocol* (SNMP) [9], desarrollado en 1988 y que mide la carga de los enlaces basándose en la medición de los paquetes y del número de bytes a través de un switch. Un *polling* (solicitud) se realiza periódicamente, siendo enviado hacia cada dispositivo IP de la red, y la información obtenida se almacena en una estructura central denominada *Management Information Base* (MIB). Sin embargo, SNMP no es un método suficientemente preciso para medir las estadísticas del nivel de flujo y el comportamiento de los hosts. NetFlow [10] y sFlow [11] presentan una solución escalable de monitorización en tiempo real basada en mediciones de flujo. Este método obtiene periódicamente información de cada flujo, dándonos una precisión aceptable, pero todavía mejorable. Las muestras tomadas son enviadas a la estructura central, espaciadas entre sí, cada cierto intervalo de tiempo. Otro método destacable es *Deep Packet Inspection* (DPI) [12], el cual nos ofrecerá estadísticas de paquetes con gran agilidad y rapidez. Su principal problema es que requerirá muchas modificaciones hardware.

3.3. Mecanismos de monitorización de tráfico con soporte de fallos de red para redes SDN

Para medir el retardo o la pérdida de paquetes en una ruta determinada, suelen realizarse medidas a nivel de aplicación. Usando paquetes ICMP (*Internet Control Message Protocol*) se podrá calcular el *round trip time* (RTT), es decir, el tiempo que invierte un paquete determinado en ir desde el nodo origen a un nodo destino y volver. Skitter [13] utiliza puntos de monitorización (balizas) para obtener el retardo y la pérdida de paquetes, pero necesita apoyarse en la instalación de infraestructura adicional. Las mediciones del retardo también podrán realizarse con métodos pasivos, aunque requieran una mayor complejidad. IPMON [14] presenta una solución que consiste en analizar las cabeceras de los paquetes TCP, y el tiempo registrado (*timestamps*) de tal medición es enviado a un servidor central para su análisis. El principal problema es que necesitará la instalación de múltiples unidades de monitorización.

Dentro de la monitorización en redes SDN, el protocolo OpenFlow permite implementar técnicas de monitorización. Trabajos como OpenTM [15] generan una matriz de tráfico (TM) para monitorizar las estadísticas de cada flujo. Dicha matriz de tráfico se va actualizando continuamente, ya que la aplicación realiza continuamente *polling* sobre los switches. OpenSAFE [16] presenta otro enfoque en la que cualquier nueva solicitud de flujo debe de pasar a través del controlador OpenFlow. El controlador enviará las reglas con la creación de nuevo flujo a los diferentes sistemas de monitorización de la red, los cuales guardarán y analizarán el tráfico medido mediante un *Intrusion Detection System* (IDS). El principal problema de este método es que requiere modificaciones hardware. OpenSketch [17] es otro trabajo basado en monitorización de redes SDN, sin embargo, para su aplicación se requiere una actualización o remplazamiento de todos los nodos de la red, ocasionando con ello una gran inversión que los proveedores de servicios de Internet se mostrarán reacios a asumir.

3.3. Mecanismos de monitorización de tráfico con soporte de fallos de red para redes SDN

3.3.1. Introducción

La arquitectura de las redes SDN, con un plano de control lógicamente centralizado y la definición de protocolos como OpenFlow, han facilitado la implementación de mecanismos de monitorización que permiten medir continuamente el rendimiento de la red. Uno de los principales objetivos de la monitorización del tráfico en redes SDN es poder proporcionar mediciones por flujo. Para ello es necesario identificar unívocamente la ruta de red completa que siguen todos y cada uno de los paquetes del flujo monitorizado.

OpenNetMon [18] propone un módulo de monitorización por flujos asociado al controlador SDN POX [19] que permite obtener métricas de *throughput* y *delay*. El mecanismo de monitorización propuesto hace uso de las características del protocolo OpenFlow, que permiten al controlador solicitar, a los dispositivos de red, las estadísticas asociadas a cada uno de los flujos que tengan definidos (mediante el uso de mensajes OFPT_MULTIPART_REQUEST) y además permite inyectar paquetes en la red (mediante el envío

de mensajes OFPT_PACKET_OUT). Para poder determinar el *throughput* de cada flujo, OpenNetMon consulta regularmente los switches para obtener las estadísticas de los flujos que tiene definidos en sus *flow tables*. Como respuesta a la solicitud, el controlador recibirá la cantidad de bytes enviados y la duración de cada flujo, lo que le permitirá calcular el *throughput* asociado al flujo. Por otro lado, para determinar el retardo total de un flujo monitorizado, el módulo inyecta regularmente paquetes de prueba a través del primer switch, de modo que estos paquetes viajan exactamente por la misma ruta que se desea monitorizar hasta el último switch, que los enviará al controlador. El controlador podrá estimar, de ese modo, el retardo completo del flujo monitorizado calculando la diferencia entre los tiempos de salida y llegada de un paquete inyectado y restando los retardos existentes entre el switch inicial y el switch final de la ruta y el controlador. El retardo entre un switch y el controlador SDN se estima a partir del RTT entre ambos, que se mide inyectando paquetes que serán directamente devueltos al controlador.

De forma resumida, OpenNetMon se fundamenta en el uso de dos módulos: el módulo de reenvío (*forwarding*) y el módulo de monitorización (*monitoring*). El primero de ellos, el módulo de *forwarding*, es el encargado de instalar las entradas en las *flow tables* de los dispositivos de red para permitir el encaminamiento adecuado del tráfico generado en la red. Además, el módulo de *forwarding* informará al módulo *monitoring* cada vez que se establezca una nueva entrada en un switch, de manera que conocerá los switches a los que deberá solicitar estadísticos para poder calcular el *throughput* de un cierto flujo y los switches a los que deberá inyectar tráfico adicional para medir el retardo. Si bien la solución propuesta ofrece una correcta monitorización tras el establecimiento y configuración de un flujo de tráfico en la red, OpenNetMon no considera situaciones de fallo en la red (en switches o enlaces). En estos casos, la solución de monitorización debería identificar el problema, reconfigurar la red y actualizar su funcionamiento para seguir monitorizando la nueva ruta asociada al flujo en cuestión.

En el trabajo [20] ampliamos la funcionalidad de OpenNetMon proponiendo un mecanismo que ofrece soporte y respuesta frente a fallos de la red. Aunque en la mayor parte del trabajo de esta tesis empleamos el controlador SDN Ryu [21], en este trabajo en particular utilizamos POX, por ser el controlador elegido por los creadores de OpenNetMon.

3.3.2. Propuesta de mejora de OpenNetMon

En este punto vamos a detallar la implementación de los mecanismos que han permitido ampliar la funcionalidad de OpenNetMon, adaptando la función de monitorización a situaciones de fallos en la red.

Como define OpenNetMon, el controlador monitoriza periódicamente las estadísticas asociadas a cada flujo de los dispositivos, gracias al envío de mensajes OFPT_MULTIPART_REQUEST, por lo que es capaz de revelar si un switch ha detectado un fallo en alguno de sus enlaces.

En ese momento, a partir de la información de *forwarding*, el controlador identifica

3.3. Mecanismos de monitorización de tráfico con soporte de fallos de red para redes SDN

los flujos que deben ser encaminados por otras rutas diferentes y elimina las entradas de las *flow table* que han dejado de ser válidas mediante la utilización de mensajes *OFPT_FLOW_MOD*. Tras lo que denominamos 'limpieza de entradas', cuando un switch reciba un paquete perteneciente a uno de los flujos eliminados de la *flow table*, solicitará al controlador qué hacer con el paquete mediante el envío de un mensaje *OFPT_PACKET_IN*. Con ello, el módulo de *forwarding* del controlador buscará y configurará una ruta alternativa para el flujo.

Además de lo anterior, proponemos una modificación de la estrategia de inyección de paquetes que implementa OpenNetMon para medir el retardo. En nuestra propuesta, los paquetes de prueba contendrán un identificador que permitirá asociar la medida del retardo entre el controlador y el switch extremo con el retardo entre switches, para así calcular el retardo total de la ruta. En la propuesta original de OpenNetMon, los paquetes de prueba se identificarán mediante un valor fijo de 253 en el campo *nw_proto*. Sin embargo, con esta configuración puede ocurrir que el paquete de prueba 'n' no llegue al switch destino, y sea el paquete de prueba 'n+1' el siguiente en ser recibido, por lo que el retrado medido asociado al paquete 'n+1' será erróneo.

Para evitar este comportamiento anómalo en OpenNetMon, decidimos añadir un identificador numérico único para cada paquete de prueba transmitido, que permite asegurar que se está midiendo el retardo correctamente.

Por último, con el objetivo de poder monitorizar si el tráfico ha sido encaminado por la ruta inicial o una ruta alternativa, el controlador mantendrá información sobre qué switches han sufrido algún fallo o siguen activos sin ninguna situación problemática.

3.3.3. Evaluación de la solución propuesta

En esta sección vamos a evaluar la propuesta de mejora de OpenNetMon descrita en la sección anterior. Para llevar a cabo esta evaluación realizaremos diferentes experimentos empleando el emulador de redes Mininet.

Figura 3.1: Configuración de una red SDN basada en la tecnología OpenNetMon

La figura 3.1 muestra la topología red empleada. Como se puede observar, la red SDN monitorizada está compuesta por seis dispositivos de red OpenFlow, en particular OpenvSwitch (OVS) [22], dos terminales extremos y el controlador POX.

Los experimentos realizados en este trabajo consistieron en la transmisión de un archivo de video entre los dos terminales de host1 y host2 usando la aplicación VLC streaming [23]. El objetivo consistió en medir los parámetros que monitoriza OpenNetMon: el retardo y el *throughput*, ambos indudablemente asociados a la calidad de la transmisión. Con el fin de evaluar la respuesta de nuestra propuesta, nos centraremos en la evaluación de ambos parámetros cuando, durante la transmisión, uno de los switches es apagado forzosamente. Por lo tanto, ante esta situación, el controlador deberá ser capaz de gestionar una ruta alternativa que permita que continúe la transmisión y su monitorización.

Figura 3.2: Detalle de la ruta principal entre host1 y host2

La figura 3.2 muestra la ruta inicial establecida por el controlador entre host1 y host2. Representada por una línea punteada, corresponde con la ruta de menor número de saltos. La figura 3.3 muestra (a) el retardo y (b) el *throughput* medido durante la transmisión de video entre host1 y host2, siguiendo el tráfico la ruta inicial, es decir, la ruta de menor número de saltos. Observamos que, si bien ambos parámetros varían con el tiempo, la media de ellos es bastante estable, 0.4098 ms y 12.2 Kbps, respectivamente.

3.3. Mecanismos de monitorización de tráfico con soporte de fallos de red para redes SDN

Figura 3.3: Retardo y *throughput* correspondiente a la transmisión de video entre host1 y host2 a través de la ruta inicial

Para evaluar la respuesta de la herramienta de monitorización frente a fallos en la red, realizaremos el mismo experimento forzando la caída del switch 2 tras 200 segundos del inicio de la transmisión.

Figura 3.4: Detalle de la ruta alternativa punteada

La ruta alternativa que se empleará corresponde a la siguiente ruta de menor número de saltos, mostrada en la figura 3.4. La ruta alternativa es la utilizada por el controlador para el envío *streaming* entre los dos hosts. La figura 3.5 muestra (a) el retardo y (b) el *throughput* medidos en este segundo experimento. Por tanto, en la figura 3.5 (a) identificamos dos regiones relevantes:

- la primera corresponde al tiempo transcurrido desde el inicio de la transmisión hasta el instante en el que el switch 2 sufre la caída (al cabo de 200 segundos), es decir, corresponde a la transmisión por la ruta principal,

- la segunda, abarca desde el segundo 200 hasta el final de la transmisión. Para forzar la caída del switch 2, ejecutamos el comando de Mininet *switch [switch number] stop*. Tras esto, el mecanismo de detección y soporte de fallos de red implementado es capaz de identificar la situación y configurar la ruta alternativa para que continúe la transmisión. Una vez configurada la ruta alternativa el valor del retardo aumenta, estabilizándose entorno a un valor medio mayor que en la primera región.

Figura 3.5: Retardo y *throughput* correspondientes a la transmisión de vídeo entre host1 y host2 en un escenario en el que switch 2 falla en el segundo 200

Tras observar los resultados obtenidos, podemos confirmar que la mejora propuesta para OpenNetMon ha sido capaz de reaccionar ante el fallo en la red, reencaminando el tráfico por una ruta alternativa. La figura 3.5 (b) representa el valor del *throughput* monitorizado durante todo el experimento. Observamos que no se aprecia variación entre los valores de *throughput* durante la transmisión por la ruta principal y el obtenido tras el fallo de switch 2, correspondiente a la transmisión por la ruta alternativa. En este caso, puesto que no hay más tráfico en la red, la selección de una ruta u otra no tiene una influencia directa sobre el *throughput*.

3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow

3.4.1. Introducción

Con el fin de adoptar las decisiones correctas desde el plano de control para así poder lograr un óptimo funcionamiento de los dispositivos de red, cualquier modificación que

3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow

se haya producido en la topología de red deberá ser actualizada. Éste es el principal motivo por el que el descubrimiento de la red, junto con su monitorización y por supuesto la ingeniería del tráfico cursado (TE), son las tareas que requerirán una mayor precisión en la gestión de redes SDN [24].

Hoy en día se dispone de gran variedad de controladores para las redes SDN como POX [19], Ryu [21], Floodlight [25] u OpenDayLight [26]. Todos ellos implementan un mecanismo de descubrimiento de la topología de red que normalmente está limitado a los elementos de red (switches y routers). La motivación para desarrollar el trabajo descrito en esta sección es que, hasta donde llega nuestro conocimiento, no existe ningún mecanismo que permita determinar la existencia de los distintos hosts en la red sin necesidad de que generen tráfico. El hecho de que las soluciones existentes sólo identifiquen a los hosts de la red una vez que han comenzado a generar tráfico es, en definitiva, un inconveniente, pues el conocimiento, a priori, de los posibles hosts supondrá una gran ventaja.

En el trabajo descrito en esta sección proponemos un mecanismo de descubrimiento de hosts para redes SDN. Este mecanismo complementa al mecanismo de descubrimiento de topología de red implementado en la mayoría de controladores SDN compatibles con OpenFlow, un mecanismo no estandarizado, basado en el uso de paquetes LLDP (*Link Layer Discovery Protocol*) [27].

3.4.2. Descubrimiento de la topología en redes OpenFlow SDN

Antes de comenzar con la descripción del mecanismo de descubrimiento de la topología en redes SDN, realizaremos una descripción del proceso de establecimiento de la conexión entre los switches y el controlador SDN.

Tras el proceso de arranque de los dispositivos, cada uno de los elementos de red y el controlador inician un proceso de establecimiento de conexión, en concreto, la denominada conexión de control. Cada switch establece una conexión TCP mediante la IP y el puerto determinado por el controlador, conexión que puede ser encriptada usando el protocolo TLS. A continuación se realiza un intercambio de mensajes OFPT_HELLO entre el controlador y los diferentes switches, mediante los cuales se pondrán de acuerdo en la versión de OpenFlow a utilizar, la más actualizada que tengan ambos disponible.

Una vez se ponen de acuerdo en la versión de OpenFlow a utilizar, el controlador enviará al dispositivo de red un mensaje OFPT_FEATURES_REQUEST. El switch responderá con un mensaje OFPT_FEATURES_REPLY el cual contendrá: el identificador único del switch (llamado *datapath_id*), los puertos que el switch tiene activos y las correspondientes direcciones MAC. Tras este intercambio de mensajes entre el controlador y los switches, el controlador es capaz de descubrir y determinar la existencia de los switches en la red, pero no las conexiones existentes entre ellos.

La figura 3.6 muestra el intercambio de mensajes descrito y también los mensajes que

se intercambiarán en el proceso de descubrimiento de conexiones entre dispositivos. La mayoría de los controladores SDN conocidos implementan un mecanismo similar basado en el uso de mensajes LLDP, procedimiento que aún no ha sido estandarizado.

Figura 3.6: Proceso de descubrimiento de topología en redes SDN basadas en OpenFlow

LLDP es un protocolo a nivel de enlace, independiente del proveedor, utilizado por los dispositivos de red para comunicar su identidad, su capacidad y sus dispositivos vecinos en redes de área local, especialmente en Ethernet por cable. Los paquetes LLDP son enviados periódicamente por los dispositivos desde cada uno de sus puertos. Dichos paquetes son encapsulados en tramas Ethernet con el campo ethertype a valor 0x88cc. Las tramas LLDP son enviadas a la dirección MAC multicast 01 : 80 : C2 : 00 : 00 : 0E. El protocolo LLDP es un protocolo de un solo salto, es decir, los paquetes LLDP sólo son recibidos por los dispositivos de red conectados directamente.

En caso de los switches OpenFlow, el envío de paquetes LLDP para el descubrimiento de dispositivos vecinos no se inicia automáticamente, sino que será el controlador, por medio de un mensaje OFPT_PACKET_OUT, el que fuerce al switch a enviar los paquetes LLDP por cada uno de sus puertos (paso 4 de la figura 3.6). Gracias a la recepción de tramas LLDP, los switches serán capaces de conocer sus dispositivos vecinos, pero dicha información no es recibida por el controlador. Para que el controlador también la conozca, se deberá añadir una entrada en la *flow table* de los switches (OFPT_FLOW_MOD), que provocará que cualquier mensaje LLDP recibido en el switch sea reenviado al controlador mediante un mensaje OFPT_PACKET_IN (paso 6 de la figura 3.6).

Como ya hemos indicado anteriormente, el mecanismo de la topología de red incluye el conocimiento de sus switches y de sus enlaces, pero no la localización de los hosts. Algunos controladores, como Ryu y OpenDayLight, implementan adicionalmente un mecanismo básico de localización de los hosts, basado en el uso de la entrada por defecto de la *flow table* (*table-miss flow entry*). Esta entrada fuerza el envío de aquellos paquetes para los que no se produzca una coincidencia al controlador. Así, cuando un host inicia la generación de tráfico y el switch al que está conectado todavía no tiene instalada una

3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow

regla útil para dicho tráfico, el paquete inicial será enviado al controlador, lo que le permitirá conocer la existencia de dicho host. En Ryu podemos encontrar la funcionalidad de topología de red en el módulo: `/ryu/topology/switches.py`.

3.4.3. Descubrimiento de hosts en redes OpenFlow

3.4.3.1. Consideraciones técnicas previas

La principal novedad y contribución del mecanismo que aquí se propone es poder descubrir la localización de los hosts sin tener que esperar a que generen tráfico. Para lograr dicho objetivo, la solución propuesta hará uso de tres tipos de información que el controlador SDN tiene a cerca de los elementos de red: los puertos del dispositivo, el estado (y los cambios de estado) de los puertos, y los enlaces entre switches.

Para conocer el estado de los puertos de un switch una vez que la conexión ya está establecida, el controlador le enviará un mensaje `OFFMP_PORT_DESC` al switch en cuestión preguntando por todos los puertos OpenFlow de dicho switch. El switch responderá al controlador facilitándole la información solicitada. Entre los parámetros que el switch emplea para describir cada uno de sus puertos, destacamos los siguientes: un identificador único de puerto, la dirección MAC del puerto, y dos campos denominados *config* y *state* compuestos por varios flags o bits bandera, cuyos significados se detalla en la figura 3.7.

```
enum ofp_port_config {  
    OFFPC_PORT_DOWN=1<<0 ;  
    /*Port is administratively down*/  
    OFFPC_NO_RECV=1<<2 ;  
    /*Drop all the pkts received by port*/  
    OFFPC_NO_FWD=1<<5 ;  
    /*Drop packets forward by port */  
    OFFPC_NO_PACKET_IN=1<<6 ;  
    /*Do not send packetin msgs for port*/  
};
```

```
enum ofp_port_state {  
    OFFPS_LINK_DOWN=1<<0 ;  
    /*No physical link present*/  
    OFFPS_BLOCKED=1<<1 ;  
    /*Port is blocked*/  
    OFFPS_LIVE=1<<2 ;  
    /*Live for Fast Failover Group*/  
};
```

Figura 3.7: Estructura de los campos *config* y *state*

El valor del campo *config* lo establece el controlador y no será modificado por el switch. El campo *state* indica el estado en el que se encuentra el enlace físico asociado a dicho puerto. Los valores de este campo son solo de lectura y no pueden ser modificados por el controlador. Según la especificación de OpenFlow 1.3, el flag *OFPP_LINK_DOWN* indica si existe enlace (flag a 1) o no (flag a 0). Cuando el valor de alguno de los flags del campo *state* cambia, el switch informará inmediatamente al controlador mediante el envío de un mensaje *OFPT_PORT_STATUS*. Sin embargo, tras la realización de diferentes pruebas en la plataforma *ONetSwitch*, pudimos comprobar que el valor del campo *state* siempre era independiente de si realmente existe un enlace físico o no cuando se recibe la solicitud de descripción de puerto. En concreto, el valor 4 se da cuando el flag *live* es 1 y los flags *blocked* y *link down* son 0. Incluso si un puerto del switch está libre y se conecta un host, el valor del campo *state* del mensaje *OFPT_PORT_STATUS* generado para informar del cambio no varía. De esta experimentación pudimos concluir que el valor del flag *link down* no nos indicará si un enlace está presente o no. En realidad, el flag *link down* es utilizado para indicar que ha ocurrido un cambio en el estado del enlace conectado al puerto, cuando dicho enlace deja de estar disponible.

Por todo ello, para implementar el módulo de descubrimiento del host, no es suficiente conocer los edge ports ni chequear el valor inicial del campo *state* indicado por la descripción que viene en el mensaje *OFPT_FEATURES_REPLY*. Es necesario implementar un mecanismo para identificar qué puertos están conectados al host. Como veremos más tarde, el mecanismo propuesto está basado en la herramienta de monitorización de redes de código abierto *nmap* [28].

3.4.3.2. Búsqueda proactiva de hosts

El modelo de descubrimiento de host propuesto deberá ser capaz de localizar los hosts que estén conectados inicialmente a la red, así como los nuevos hosts que se puedan unir a la red durante el período de tiempo en el que esté siendo monitorizada.

El primer objetivo del mecanismo propuesto es que el controlador sea capaz de diferenciar entre puertos del switch que no están conectados y puertos que están conectados a hosts. Para dar respuesta a este objetivo, implementaremos un mecanismo inspirado en la opción de detección de hosts de la herramienta *nmap*.

Tras el establecimiento de la sesión, y una vez que se hayan intercambiado los mensajes de descripción de puerto, el controlador iniciará una búsqueda proactiva de hosts, cuyo funcionamiento es descrito en el pseudocódigo de la tabla 3.1.

Durante el proceso de búsqueda, el controlador enviará un conjunto de mensajes *OFPT_PACKET_OUT* a cada switch que se encuentre en los extremos de la topología (edge switch). Cada uno de los mensajes contendrá un paquete *ARP_Request* que intenta resolver cada una de las direcciones IP del conjunto de direcciones pertenecientes al rango asignado a la red. Aunque el paquete *ARP_Request* lo ha generado el controlador, la

3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow

dirección MAC origen utilizada será la dirección física del *edge port* del switch. De esta forma, el switch podrá diferenciar entre un paquete *ARP_Response* generado por un host ante la recepción del *ARP_Request* generado proactivamente, y un *ARP_Response* asociado a tráfico normal, es decir, asociado a la generación de tráfico entre hosts existentes. En el primer caso, el paquete *ARP_Response* será reenviado al controlador.

```
1: proactive_host_discovery:
2:   for switch in network.switches:
3:     for port in switch.ports:
4:       if edge_port(port):
5:         sending_ping_arp_flowmod(switch,port)
6:         sending_ping_arp(switch,port,netaddress)
7:
8: sending_ping_arp_flowmod(switch,port):
9:   match = (ether_types==ARP & eth_dst==port.hw_addr)
10:  action = output_to_controller
11:  msg = create_flow_mod_message(match,action)
12:  send(msg,switch)
13:
14: sending_ping_arp(switch,port,netaddress):
15:   for ip in range netaddress:
16:     dst = FF:FF:FF:FF:FF:FF
17:     src = port.hw_addr
18:     dst = ip
19:     src_ip = broadcast_netaddress
20:     ethfr = ethfr_frame(ARP_request,dst,src,dst,src,dst_ip,src_ip)
21:     send_packet_out(ethfr,switch,port)
```

Tabla 3.1: Descubrimiento proactivo de host. Pseudocódigo

Cuando un switch reciba cada uno de los mensajes OFPT_PACKET_OUT desde el controlador, enviará un *ARP_Request* encapsulado por el puerto correspondiente. Si en dicho puerto no hay ningún host conectado, no se generará ningún *ARP_Response*. En el caso contrario, sí se generará un paquete *ARP_Request*, que será recibido por el switch. Sin embargo, el objetivo es que el controlador reciba también este paquete, para que obtenga una visión total y completa de la red. Para permitirlo, el controlador enviará un mensaje OFPT_FLOW_MOD a cada switch (*edge switch*), antes de enviarle el mensaje OFPT_PACKET_OUT. Con este primer mensaje OFPT_FLOW_MOD, el controlador incluirá una nueva entrada en la *flow table* de cada switch que identificará (hará *matching*) aquellos paquetes ARP cuya dirección MAC destino es la dirección MAC del puerto edge en cuestión. La acción asociada a esta entrada será el envío del paquete al controlador.

Una vez reciba los *ARP_Response* generados como respuesta a los *ARP_Request* generados proactivamente, el controlador los procesará como tráfico ARP normal, tal y como

está definido en el módulo de descubrimiento de topología inicial (en el caso del controlador Ryu en `/topology/switches.py`), y consecuentemente, el host asociado será incluido en la lista de hosts mantenida por el controlador.

3.4.3.3. Seguimiento de las conexiones de los hosts

Está claro que conocer el posicionamiento de los distintos hosts en la topología de red es una gran ventaja a la hora de monitorizarla. Sin embargo, otra cuestión que deberemos tener en consideración es el dinamismo que pueda presentar la propia red. Es decir, tener presente que los puertos pueden desconectarse, que los enlaces pueden sufrir algún tipo de fallo, y también que nuevos hosts pueden ser añadidos a la misma. Por lo tanto, el mecanismo de descubrimiento de los hosts propuesto también deberá saber adaptarse y reaccionar ante los cambios en la red que afecten a las conexiones de los hosts.

Como describimos anteriormente, el cambio del estado de un puerto de un switch será notificado al controlador mediante un mensaje `OFPT_PORT_STATUS`. Los motivos del envío de un mensaje de este tipo son: que un puerto ha sido añadido (contendrá la opción `OFPPR_ADD`), que un puerto ha sido eliminado (contendrá la opción `OFPPR_DELETE`), o que un puerto ha modificado su estado (contendrá la opción `OFPPR_MODIFY`).

Los dos primeros mensajes permitirán al controlador mantener actualizado los puertos involucrados en la topología de red. Por el contrario, el tercer tipo de mensaje involucra situaciones algo más complejas.

Una de estas situaciones correspondería a la desconexión inesperada de un enlace. Si un enlace se desconecta repentinamente, el switch detecta que ya no existe un enlace físico y entonces se envía un mensaje `OFPT_PORT_STATUS` al controlador indicando el cambio en el estado del puerto de 4 (el flag *live* puesto a 1) a 5 (tanto el flag *live* como el flag *link down* puestos a 1).

Para profundizar en el estudio de este evento, asociado a la desconexión física de un enlace, realizamos una serie de experimentos. En un primer experimento empleamos un `ONetSwitch` que tiene cuatro puertos no conectados. Mediante un mensaje `OFPT_MULTIPART_REQUEST` del tipo `OFPPM_PORT_DESC`, el switch informa al controlador del estado de los puertos (en este caso el valor que presenta el campo *state* del mensaje es 4, ya que solo el flag *live* está activo). En un momento determinado, un host (que está encendido) se conecta al switch. En esta situación, aunque un nuevo enlace físico ha sido creado entre ellos, no se genera ningún mensaje OpenFlow indicando el cambio en el estado del puerto. Como ya verificamos anteriormente, a pesar de la definición que da el estándar del bit *link down*, este flag sólo se emplea para indicar que un enlace existente deja de estar disponible, y no para indicar que un enlace que no estaba activo pasa a estarlo. Si el host activo es físicamente desconectado del switch, un mensaje `OFPT_PORT_STATUS` generado por la razón `OFPPR_MODIFY` es enviado por el switch hacia el controlador, en este caso el valor del campo *state* a 5, ya que los flags *live* y *link down* están activados. Si volvemos a conectar el switch, esta vez un mensaje `OFPT_PORT_STATUS` del tipo

3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow

OFPPR_MODIFY es enviado hacia el controlador con valor 4 en el campo *state*. Finalmente podemos corroborar la conclusión anterior, el flag *link down* está relacionado con la caída de un enlace físico, pero no con el establecimiento de un nuevo enlace.

En un segundo experimento partimos desde el mismo escenario, pero esta vez tenemos tres puertos sin conexión y el cuarto puerto está conectado a un host apagado. Igual que en el primer experimento, un mensaje OFPT_MULTIPART_REQUEST del tipo OFPMP_PORT_DESC es enviado al controlador con valor 4 en el campo *state* asociado al puerto al que se conecta el host apagado (solo el flag *live* activado). Una vez que el host se enciende, el switch envía una serie de mensajes OFPT_PORT_STATUS con la opción OFPPR_MODIFY hacia el controlador. Tras capturar y analizar estos mensajes, concluimos que han sido generados debido a una fase de autonegociación de parámetros Ethernet. El valor inicial del *port state* que el switch envió al controlador era 4. Entonces, cuando se activa el host, el switch indica al controlador que el enlace cambia (se envía un mensaje OFPT_PORT_STATUS con la opción OFPPR_MODIFY con el valor *port state* 5) hasta que finaliza la fase de negociación de 100 Mbps. A continuación, se envía un nuevo mensaje con OFPT_PORT_STATUS del tipo OFPPR_MODIFY con el valor *port state* 4. Seguidamente, el switch indica al controlador que el enlace cambia (un mensaje con OFPPR_MODIFY con el valor *port state* 5 es enviado) hasta que finaliza la fase de negociación de 1000 Mbps. En este momento, un nuevo mensaje OFPT_PORT_STATUS con OFPPR_MODIFY con el valor *port state* 4 es enviado. Por último, cuando el host se desconecta de nuevo, se envían dos mensajes OFPT_PORT_STATUS al controlador: el primero indicando un valor *port state* de 5 y el segundo un valor de *port state* de 4.

Teniendo en cuenta las conclusiones de los experimentos anteriores, el módulo de descubrimiento de hosts implementado en el controlador debe reaccionar a los mensajes OFPT_PORT_STATUS generados por la razón OFPPR_MODIFY. Si el valor recibido de *port state* es 5, deberá revisar si previamente había conectado un host a dicho puerto. Si es así, el host deberá ser eliminado de la lista de hosts activos en la red. Si el valor recibido de *port state* es 4, este puerto debería ser revisado solo en el caso de que se tratase de un edge port. En este caso deberá iniciarse una búsqueda proactiva de host en el puerto en cuestión. Sin embargo, debido a que la unión del host que está conectado no genera un cambio en el estado de puerto y por tanto no se generará ningún mensaje para ser enviado al controlador, es necesario repetir periódicamente el mecanismo proactivo en todos los puertos edge.

3.4.4. Evaluación del mecanismo de descubrimiento de hosts

Usando el mecanismo basado en el intercambio de mensajes LLDP, los controladores SDN pueden descubrir los switches de la red y los enlaces entre ellos. Esta información es útil a la hora de descubrir la ruta más corta entre un puerto del switch origen y otro puerto del switch destino en la topología de la red. Sin embargo los controladores SDN sólo realizan mecanismos de detección de hosts básicos, siempre condicionados a la generación de tráfico por parte de los hosts.

La mejora del mecanismo de descubrimiento de la topología de red, incluyendo el descubrimiento avanzado de hosts, puede ser de gran utilidad en las redes SDN. Por ejemplo, en los data-centers el procedimiento para mantener las ubicaciones de las máquinas virtuales es tedioso y propenso a errores debido a su frecuente migración [29]. En este escenario, la implementación del módulo de descubrimiento de host, junto con la adecuada herramienta de monitorización de la red, nos ofrecerían una manera sencilla de gestionar la red dinámicamente optimizando la utilización de los recursos de red.

Por otro lado, la utilización del módulo de descubrimiento de hosts puede ser útil para solucionar el problema conocido como *ARP flooding* o inundación de tráfico ARP. Con la solución propuesta en este trabajo solo se envía tráfico ARP a los switches extremos (edge).

Por último, vamos a realizar una serie de experimentos para mostrar la utilidad de la solución propuesta, que permite la identificación de hosts para ser empleada en la gestión de la red. Para ello consideraremos una red *fat-tree* (figura 3.8), topología ampliamente utilizada en data-centers. En estos experimentos emplearemos la herramienta de emulación de redes Mininet.

Figura 3.8: Topología Fat-tree

3.4. Detección de hosts: mejora del descubrimiento de topologías de red SDN OpenFlow

En este escenario de pruebas creamos un flujo UDP entre el host h2 (10.0.0.2) y el host h5 (10.0.0.5) utilizando la herramienta *iperf*. Cuando el switch s2 reciba el primer paquete UDP procederá de h2 observará que en su *flow table* no hay ninguna entrada adecuada para el procesamiento de ese flujo. Por tanto, se producirá un evento *miss-table*, enviándose el paquete al controlador mediante un mensaje OFPT_PACKET_IN. El controlador, que conoce la posición de h5 gracias a la funcionalidad de descubrimiento de hosts, estará capacitado para calcular la ruta óptima entre h2 y h5, y con ello incluir en cada uno de los switches involucrados una entrada adecuada en su *flow table*. Si en un momento dado h2 fallase o migrase a otro punto de la red, es útil que el controlador detecte dicho cambio, de un modo inmediato, para así poder actuar en consecuencia.

En nuestro test, la duración de la conexión UDP fue de 180 segundos. Durante ese periodo de tiempo h5 sufrió tres desconexiones y conexiones forzadas. En el segundo 30, h5 se desconecta de la red durante 30 segundos; en el segundo 90, h5 se desconecta la red durante 15 segundos y; en el segundo 120, h5 se desconecta la red durante 30 segundos. Gracias al módulo de descubrimiento de hosts, cada vez que h5 se desconectó, el controlador fue capaz de identificar el cambio y, de forma inmediata, eliminar las entradas correspondientes en los switches de la ruta creada entre h2 y h5.

La figura 3.9 muestra el flujo entrante asociado al tráfico UDP entre h2 y h5. El tráfico UDP fue monitorizado gracias a una herramienta de monitorización presentada en la sección anterior y publicada en [20]. El flujo ha sido monitorizado en el último switch de la ruta, ya que éste es lo que ve el receptor. Podemos observar que no hay flujo durante los intervalos de tiempo en los que no está conectado h5. Resultados similares se obtienen cuando el resto de los switches son monitorizados.

Figura 3.9: Medición del ancho de banda de flujo obtenida en el último switch s5

OpenFlow también ofrece la posibilidad de obtener estadísticas detalladas sobre puertos específicos en switches seleccionados. Haciendo uso de dicha funcionalidad, obtuvimos los parámetros mostrados en la figura 3.10. El primer gráfico muestra la velocidad de transmisión de los puertos en el switch s2 (es decir, el primer switch en el camino entre h2 y h5). El segundo gráfico corresponde al switch s5, el último switch de la ruta. En esta figura se puede observar la velocidad de recepción de los puertos del switch s5, y aún más descriptivo, la velocidad de transmisión del puerto al que está conectado h5. Como se puede ver, la velocidad de transmisión coincide con el resultado esperado. Los dos últimos gráficos representan las tasas de transmisión y recepción de los puertos de los switches s1 y s4. Es interesante señalar que, como se observa en estos gráficos, al principio de la comunicación, el controlador decide que el camino óptimo entre h2 y h5 es s2-s4-s5. Sin embargo, después de la primera vez que h5 se desconecta de la red y luego se conecta de nuevo, el controlador decide seleccionar una nueva ruta entre h2 y h5: s2-s1-s5. Después de la tercera y cuarta conexión a la red, el controlador elige de nuevo la ruta s2-s4-s5.

Figura 3.10: Velocidad de transmisión y recepción de los puertos de los switches s2, s5, s1 y s4

3.5. Conclusiones

Tradicionalmente, se han usado diferentes técnicas de monitorización en las redes. Dichas técnicas suelen requerir una instalación de dispositivos físicos externos o alguna actualización software. Esta manera de proceder suele ser una ardua tarea de realizar. SDN, junto con el protocolo OpenFlow, nos ofrecen un nuevo marco de trabajo con el que poder llevar a cabo la monitorización de la red de una manera más sencilla.

El protocolo OpenFlow nos ofrece diferentes herramientas para poder monitorizar la de red sin una gran modificación sobre ella. Existen diferentes tipos de técnicas de monitorización, las basadas en medidas activas, mediante la inyección de paquetes en la red, y pasivas, mediante la observación del tráfico de red sin realizar dicha inyección. Los mecanismos de monitorización requieren identificar de manera precisa la ruta de red. El elemento que posee una vision global de la red es el controlador, por lo que dicha acción suele estar vinculada a él. Sin embargo, mientras un flujo está activo, cualquiera de los switches de la ruta puede sufrir diferentes tipos de fallos. Gracias al uso del protocolo OpenFlow en las redes SDN, hemos presentado un mecanismo sencillo que mejora la solución de monitorización OpenNetMon permitiendo reaccionar ante posibles fallos en la red.

Por otro lado, se ha propuesto una solución para implementar el descubrimiento y seguimiento de hosts dinámicos en redes basadas en SDN. Teniendo en cuenta que la mayoría de los controladores SDN incluyen el protocolo LLDP como mecanismo para descubrir los nodos de red y los enlaces entre ellos, quedarían por determinar los hosts existentes. Para subsanar este problema, se ha desarrollado un método capaz de realizar el descubrimiento de hosts analizando simplemente diferentes aspectos del estándar 1.3 de OpenFlow, el flag *ports state* y la relación existente con el mensaje OFPT_PORT-STATUS.

Influencia de las redes SDN en el ahorro energético

4.1. Introducción

En los últimos años, el concepto de eficiencia energética ha ganado popularidad debido a los problemas medioambientales y también a razones económicas. El término *green* ha ido tomando mayor importancia en el entorno de las redes, llegando a convertirse en uno de los temas de referencia tanto en la industria como en la comunidad investigadora. Hoy en día, las redes de datos suelen estar diseñadas para soportar la carga máxima en lo que a volumen de tráfico se refiere. Dicha idea nos conduce a escenarios donde la mayor parte del tiempo las redes están infrautilizadas, como pueden ser los data-centers que suelen desarrollarse mediante arquitecturas de red con elementos redundantes para ofrecer un alto grado de disponibilidad. El problema de dicha redundancia radica en la baja eficiencia energética que presenta en la mayoría de los casos. Para solventar este problema, aplicaremos la estrategia de configurar los dispositivos menos utilizados en modo reposo, siempre y cuando dicha configuración no afecte el rendimiento de la red.

El impacto de Internet en la vida diaria ha aumentado indudablemente en los últimos años. Según [30], el pico global de volumen de trabajo de la red (*throughput*) se ha incrementado aproximadamente en un 40 % desde 2012. Dicho incremento de demanda de servicio está asociado a un mayor número de dispositivos conectados a la red: smartphones, tablets, ordenadores, etc.

Para poder mantener el ritmo de crecimiento previsto, los proveedores de servicios de Internet (ISP) deberán de confiar en un crecimiento similar, tanto en el ancho de banda de los enlaces como en las capacidades de los routers y switches. Ello provocará un aumento muy significativo en el consumo de energía y por tanto en la emisión de CO_2 . Aunque ahora somos conscientes de las enormes emisiones de carbono y de las necesidades energéticas, es obvio que las emisiones de carbono y la cantidad de consumo de energía seguirán aumentando [31]. Como se indica en el estudio SMART 2020 [32], las emisiones de CO_2 basadas en las TIC están aumentando a un ritmo del 6 % anual. Con

este índice de crecimiento, se espera que las emisiones de CO_2 causadas por las TIC alcancen el 12 % de las emisiones mundiales en 2020. En este escenario de aumento global de consumo de energía, los data-centers y las redes troncales (*backbone*) seguirán siendo los principales responsables de dicho incremento [33]. Por otro lado, la Agencia Internacional de la Energía (IEA) [34], publicó en el año 2015 que la electricidad utilizada por las redes de Internet se estimaba en unos 185 TWh, es decir aproximadamente un 1 % de la demanda global de electricidad, y que se estima que en el año 2021 se alcance un consumo de hasta 320 TWh.

Por todo lo expuesto anteriormente, la reducción del consumo de energía en las redes de telecomunicaciones es un paso crucial a fin de lograr ahorros significativos de energía a nivel global, así como reducir las emisiones de CO_2 . En este escenario, SDN se presenta como una solución muy adecuada para realizar un enrutamiento lo más adaptado y flexible posible, a fin de lograr reducir el consumo de energía.

En este capítulo presentamos nuestra contribución al logro de eficiencia energética a través de las redes SDN, que ha dado lugar a la siguiente publicación:

- A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez and J. Malgosa-Sanahuja. "Optimization of Power Consumption in SDN Networks". En: 2017 The Ninth International Conference on Emerging Networks and Systems Intelligence (EMERGING 2017). Nov. de 2017, págs. 12-15.

4.2. Evolución del enrutamiento energético en las redes SDN

La idea básica del concepto *Energy-Aware Routing* (EAR) consiste en administrar el tráfico entrante en la red con el objetivo de reducir el número de recursos no utilizados ajustando la capacidad ofrecida al uso real de la red como una solución potencial para la minimización del consumo de energía. Las soluciones EAR fueron propuestas inicialmente para las redes IP tradicionales. Sin embargo, las desventajas o limitaciones de implementar estos enfoques en redes tradicionales han llamado la atención de la comunidad investigadora, que propone el uso de redes SDN para minimizar el consumo de energía.

En las redes tradicionales, el fuerte acoplamiento entre los planos de datos y de control hace muy difícil el despliegue de algoritmos de enrutamiento que tengan en cuenta la energía. También puede resultar muy costoso cuando los numerosos dispositivos provienen de diferentes fabricantes, o cuando utilizan diferentes interfaces de programación o diferentes protocolos. Por el contrario, una arquitectura basada en SDN elimina estas limitaciones mediante la separación del plano de control del plano de datos, y la existencia de una entidad controladora SDN, que proporciona una arquitectura lógicamente centralizada.

Tal y como presenta Dabaghi et al. [35], los mecanismos EAR en arquitecturas tradicionales se pueden dividir en dos subgrupos:

4.2. Evolución del enrutamiento energético en las redes SDN

- enfoque que ignora el tráfico de red. Dentro de este bloque podemos destacar varios trabajos [36]-[39]. Estas propuestas logran ahorrar gran cantidad de energía, pero en su contra, hay que resaltar que pueden llegar a provocar una gran congestión en la red durante los periodos de máxima carga.
- enfoque que tiene en cuenta el tráfico de red. Este tipo de enfoque suele ser más común, ofreciendo el cumplimiento de parámetros QoS y un aceptable nivel de energía ahorrada. Este tipo de propuesta suele ser formulada como un problema *integer linear programming* (ILP) o *mixed integer linear programming* (MILP), donde el ahorro energético suele ser el objetivo de la función. Debido a que el problema se presenta como un problema NP-hard [40], es necesaria la aplicación de diversos métodos o algoritmos heurísticos.

Siguiendo el segundo enfoque, Heller et al. [41] presentan el algoritmo *ElasticTree*, una de las propuestas de ahorro energético más conocidas y utilizadas en redes data-center con tecnología SDN. La idea más novedosa que presenta este algoritmo es la desconexión tanto de enlaces como de switches de la topología de red basándose en la cantidad de carga que soporta. Los autores muestran que los flujos de tráfico presentes en la red pueden ser abastecidos por conjuntos pequeños de enlaces y switches durante la mayor parte del tiempo. En [42] se decide enfocar el problema de optimización del consumo energético en la red añadiendo nuevas variables. Para ello, se implementan más estados intermedios en los dispositivos de red aparte de los estados ON y OFF. En [43] los autores presentan un prototipo cuya principal ventaja consistirá en poder crear un modelo de gestión de la energía en tiempo real, monitorizando para ello tanto el tráfico de la red como la demanda de los usuarios. El problema se expone mediante un problema ILP gracias al cual se logra un ahorro de energía en los enlaces y los switches sin disminuirse el nivel de QoS requerido. El principal problema de este prototipo es que no se tiene en cuenta la limitación del tamaño de la *flow table* de cada uno de los switches OpenFlow, siendo ésta una de las mayores limitaciones en la práctica. Muchas de las propuestas EAR suponen que el espacio de las *flow tables* es ilimitado, por lo que Giroire et al. [44] presentan su propia propuesta teniendo en consideración el espacio limitado de las TCAM (*Ternary Content Addressable Memory*) en los dispositivos SDN. La conclusión de este tipo de arquitecturas es que gracias al uso de redes basadas en SDN podemos proponer e implementar soluciones EAR cuya lógica de red se encuentre centralizada en la figura del controlador. Los enfoques de SDN también permiten bajos gastos operativos y la flexibilidad para administrar la red y mejorar la QoS.

Los modelos energéticos que presentan más similitudes a nuestra propuesta son [45] y [46]. En [45] los autores hacen frente al problema del consumo energético de la red fijándose como objetivo lograr unos enrutamientos que minimicen el número de dispositivos activos. En [46] se presenta una continuación del modelo anterior, desarrollando un algoritmo heurístico que converge más rápido.

4.3. Optimización del consumo de energía en redes SDN

4.3.1. Introducción

Tal y como se ha comentado en apartados anteriores de este mismo Capítulo 4, la optimización de la potencia consumida en las redes de Internet está considerada a día de hoy como un campo de gran interés en la comunidad investigadora. Las altas previsiones de consumo de energía y de emisiones (de CO_2) han convertido en prioritario el objetivo de lograr adecuadas y novedosas soluciones al respecto. También hemos apuntado que la aparición e implementación de SDN ha supuesto un gran empuje en lo que a ahorro energético se refiere, ya que este tipo de redes nos ofrecen la posibilidad de tener un control directo en el comportamiento de la red pudiéndose, por tanto, interactuar sobre todos y cada uno de los diferentes dispositivos de la topología. Gracias a dicha comunicación directa entre el controlador y los dispositivos de red se nos ofrecen diversas posibilidades como: desconectar switches y enlaces, modificar la velocidad de cada enlace, configurar diferentes rutas, considerando en todo momento que cualquiera de estas acciones podrán ser ejecutadas siempre y cuando la QoS no se vea afectada.

Es sabido que el mayor ahorro energético de cualquier red se puede lograr desconectando la mayor cantidad posible de dispositivos activos. Sin embargo, el gasto energético de las redes de datos también podrá alcanzarse modificando el estado de sus elementos activos, por ejemplo, configurando elementos de red a un modo de reposo (*sleep*). Nuestro análisis y estudio podrá considerarse como una extensión de los trabajos [45] y [46].

4.3.2. Propuesta Energy-Aware

Tal y como se analiza en [47], la potencia consumida por un switch OpenFlow se puede modelar como:

$$P_{switch} = P_{base} + P_{config} + P_{control} + P_{OF} \quad (4.1)$$

La potencia total se obtiene como la suma de cuatro factores (o tipos distintos de potencia) que se detallan a continuación:

- la potencia base propia del switch, representada por P_{base} ,
- la potencia de la configuración propia del switch activo, representada por P_{config} . Esta potencia dependerá tanto del número de puertos activos como de la velocidad configurada en cada uno de ellos,
- la potencia del tráfico de control, representada por $P_{control}$. Esta potencia dependerá de la tasa de envío de mensajes Packet_In, de la tasa de recepción de mensajes FlowMod y de los respectivos consumos de energía por paquete,
- la potencia del tráfico OpenFlow y que representaremos por P_{OF} . Esta potencia dependerá tanto de las coincidencias al comparar el tráfico recibido con las entradas de la *flow table* del dispositivo (*matches*) como de las acciones de las reglas OpenFlow activas.

4.3. Optimización del consumo de energía en redes SDN

El objetivo de este trabajo es encontrar una solución que permita minimizar el consumo total de energía en la red, para lo cual buscaremos:

- obtener el número mínimo de switches que deberán activarse en la red,
- obtener el número mínimo de enlaces que deberán activarse en la red,
- definir la capacidad de cada uno de los enlaces activos,
- definir como deberá encaminarse el tráfico ofrecido a la red SDN a través de los enlaces activados en dicha red.

Nuestro modelo considera una topología de grafo unidireccional, caracterizado por $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, donde $\mathcal{N}$ son los nodos de la red, incluyendo el conjunto de hosts $\mathcal{H}$ como al conjunto de switches $\mathcal{V}$. $\mathcal{E}$ determina los enlaces existentes entre los nodos $\mathcal{N}$. Considerando $\mathcal{D}$ como el conjunto de demandas de tráfico entre cualquier par de nodos, podremos definir que a cada demanda d le corresponde un tráfico ofrecido h_d .

Para poder resolver este problema de optimización, definimos los siguientes parámetros de entrada y las siguientes variables de diseño.

Parámetros de entrada:

- $\mathcal{H}$: conjunto de hosts de la red SDN,
- $\mathcal{V}$: conjunto de switches SDN,
- $\mathcal{E}$: conjunto de enlaces de la red. A partir de esta información, $\delta^+(v)$ denota el conjunto de posibles enlaces que salen del nodo v , y $\delta^-(v)$ el conjunto de posibles enlaces entrantes a v ,
- $\mathcal{D}$: conjunto de demandas de tráfico unicast,
- $h_d, d \in \mathcal{D}$: tráfico ofrecido asociado a una demanda d . A partir de esta información, $a(d)$ indica el host origen de una demanda d y $b(d)$ el host destino,
- P_{base} : potencia base de los switches SDN,
- $P_{config-port}$: potencia asociada a la activación de los dos puertos extremos de un determinado enlace,
- $P_{config-speed}$: definida como la potencia asociada a la velocidad, del enlace activo y configurado,
- P_{OF} : potencia asociada al tráfico de control y al tráfico OpenFlow,
- U : capacidad máxima de un enlace,
- M : número auxiliar grande.

Variables de decisión:

- $z_e, e \in \mathcal{E}$:

$$\begin{cases} 1, \text{ si el enlace } e \text{ está configurado (activo)} \\ 0, \text{ en caso contrario} \end{cases}$$

- $u_e, e \in \mathcal{E}$: la capacidad definida para el enlace e ,

- $x_{de}, d \in \mathcal{D}, e \in \mathcal{E}$: la demanda de tráfico d que atraviesa el enlace e ,

- $\hat{x}_{de}, d \in \mathcal{D}, e \in \mathcal{E}$:

$$\begin{cases} 1, \text{ si la demanda } d \text{ atraviesa el enlace } e \\ 0, \text{ en caso contrario} \end{cases}$$

- $x_v, v \in \mathcal{V}$:

$$\begin{cases} 1, \text{ si el nodo } v \text{ está activado} \\ 0, \text{ en caso contrario} \end{cases}$$

- $d_v, v \in \mathcal{V}$: Numero de flujos OpenFlows instalados en el switch v .

Teniendo en cuenta lo anterior, la formulación del problema de minimización se puede expresar como:

$$\min(P_{base} \sum_n x_n + P_{config-port} \sum_e z_e + P_{config-speed} \sum_e u_e + P_{OF-routing} \sum_n d_n) \quad (4.2)$$

que debe estar sujeta a las siguientes restricciones:

$$\sum_{e \in \delta^+(v)} x_{de} - \sum_{e \in \delta^-(v)} x_{de} = \begin{cases} h_d, & \text{if } v = a(d) \\ -h_d, & \text{if } v = b(d) \\ 0, & \text{otherwise} \end{cases} \quad (4.3)$$

$$\forall d \in \mathcal{D}, \forall v \in \mathcal{V} \quad (4.4)$$

4.3. Optimización del consumo de energía en redes SDN

$$\sum_d x_{de} \leq u_e, \quad \forall e \in \mathcal{E} \quad (4.5)$$

$$u_e \leq U z_e, \quad \forall e \in \mathcal{E} \quad (4.6)$$

$$\sum_{e \in \delta^+(v)} u_e + \sum_{e \in \delta^-(v)} u_e \leq M x_v, \quad \forall v \in \mathcal{V} \quad (4.7)$$

$$x_{de} \leq M \hat{x}_{de}, \quad \forall d \in \mathcal{D}, \forall e \in \mathcal{E} \quad (4.8)$$

$$\sum_{e \in \delta^+(v)} \sum_d \hat{x}_{de} = d_v, \quad \forall v \in \mathcal{V} \quad (4.9)$$

$$u_e \geq 0, x_{de} \geq 0, d_v \geq 0, \quad \forall d \in \mathcal{D}, e \in \mathcal{E}, v \in \mathcal{V} \quad (4.10)$$

$$z_e, \hat{x}_{de}, x_v \in \{0, 1\} \quad \forall d \in \mathcal{D}, e \in \mathcal{E}, v \in \mathcal{V} \quad (4.11)$$

en las que los detalles complementarios y explicativos a cada una de ellas serán los siguientes:

- Las restricciones (4.4) representan las condiciones de conservación de flujo
- Para cada enlace activo, las restricciones (4.5) indican que el tráfico cursado por un enlace debe ser menor o igual que su capacidad
- Las restricciones (4.6) fuerzan que:
 - si un enlace e no está activo ($z_e = 0$), entonces no puede existir capacidad en él ($u_e = 0$),
 - si un posible enlace e está activo ($z_e = 1$), su capacidad estará limitada al valor de U .
- Según las restricciones (4.7), si un switch v no está activado ($x_v = 0$), entonces la capacidad de sus enlaces salientes y entrantes tiene que ser igual a cero ($u_e = 0$),
- Según las restricciones (4.8), si una demanda d no atraviesa un enlace e , entonces el tráfico asociado de la demanda tendrá que ser igual a cero ($x_{de} = 0$),
- (4.9) permite obtener el número de flujos OpenFlow instalados en el switch v ,
- Finalmente, las restricciones (4.10) prohíben que un enlace lleve una cantidad negativa de tráfico de una demanda, ya que esto no tiene ningún significado físico.

A pesar de que el modelo presentado en este apartado nos permite encontrar la solución óptima de consumo energético en redes SDN, es un problema *NP-Hard* con una costosa solución para redes tradicionales de tamaño medio-alto. Por lo que desarrollamos un algoritmo heurístico para obtener una solución de una forma menos compleja.

4.3.3. Algoritmos Heurísticos

Para resolver el problema descrito en la sección anterior, hemos propuesto el algoritmo heurístico presentado en la tabla 4.1. Podemos observar que la labor del bucle principal del algoritmo consistirá en definir y determinar la ruta con menor consumo de potencia para cada posible demanda. En la *línea 2* de dicho algoritmo, observamos que para la primera demanda de tráfico se elegirá la ruta de camino más corto [48]. Una vez establecido dicho camino, se procederá al cálculo del consumo de energía asociado a la configuración y el establecimiento de dicha ruta, actualizándose los conjuntos de switches activos ($\mathcal{V}$), enlaces activos ($\mathcal{X}$), capacidades de los enlaces ($\mathcal{U}$) y flujos en cada conmutador ($\mathcal{F}$).

Require: $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ network graph, $\mathcal{D}$ data traffic demands

Ensure: $\mathcal{V}$ active switches, $\mathcal{X}$ active links, $\mathcal{U}$ links capacity, $\mathcal{F}$ flows in each switch

```

1: for all demands  $\in \mathcal{D}$  do
2:   if first demand then
3:     path=SHORTESTPATH(demand)
4:     CALCULATEPOWER(path, demand)
5:     UPDATE( $\mathcal{V}, \mathcal{X}, \mathcal{U}, \mathcal{F}$ )
6:   else
7:      $\mathcal{P}$ =ALLPOSSIBLEPATHS(demand)
8:     for all paths  $\in \mathcal{P}$  do
9:       if AVAILABLECAPACITY(path) then
10:        power=CALCULATEPOWER(path, demand)
11:      end if
12:    end for
13:    SELECTMINIMUMPOWERPATH( $\mathcal{P}$ )
14:    UPDATE( $\mathcal{V}, \mathcal{X}, \mathcal{U}, \mathcal{F}$ )
15:  end if
16: end for

```

Tabla 4.1: Algoritmo heurístico propuesto

Para el resto de demandas de tráfico (*línea 6*), en primer lugar se obtendrá el conjunto de los posibles encaminamientos asociados a dicha demanda. Entre todas las rutas posibles con capacidad suficiente para establecer la nueva conexión en los enlaces implicados, se elegirá la ruta que ocasione un menor incremento de consumo de energía en la red, como podemos observar en la *línea 13*. Una vez seleccionada la ruta óptima, se procederá a la actualización de los parámetros significativos: switches activos ($\mathcal{V}$), los enlaces activos ($\mathcal{X}$), la capacidad de los enlaces ($\mathcal{U}$) y los flujos de cada interruptor ($\mathcal{F}$).

Para obtener el incremento de energía total que supone el establecimiento de determinada ruta para dar respuesta a una determinada demanda, utilizaremos el método mostrado en la tabla 4.2. La *línea 2* analiza si deberá activarse un nuevo enlace o si habrá que incrementar la velocidad configurada, en el caso de que ya estuviese activado. La *línea 3* analiza y determina la necesidad de incluir nuevas entradas de flujo en los distintos swit-

4.3. Optimización del consumo de energía en redes SDN

ches de la ruta. La *línea 5* determina si es necesario activar un switch que estaba apagado para poder establecer la ruta. Finalmente, tras las actuaciones antes descritas, la *línea 7* actualiza el que sería el nuevo coste total de energía si se utilizara la ruta considerada.

```
1: for all links  $\in$  path do
2:   UPDATEACTIVELINKS(link, demand)
3:   UPDATEINSTALLEDFlOWS(demand)
4:   if origin(link) or destination(link)  $\notin \mathcal{V}$  then
5:     UPDATEACTIVESWITCHES( )
6:   end if
7:   UPDATETOTALPOWER(demand)
8: end for
9: return totalPower
```

Tabla 4.2: CALCULATE POWER (path,demand)

A fin de poder evaluar el rendimiento de nuestro algoritmo en distintos escenarios, se ha desarrollado un simulador en lenguaje Python que hace uso del paquete NetworkX para la creación, estudio y configuración de diferentes topologías de red. En nuestro caso, se realizaron dos estudios, con dos topologías distintas. La primera topología corresponde a una red de veinte switches $|\mathcal{V}| = 20$ y la segunda a una red de $|\mathcal{V}| = 70$. Las demandas de tráfico empleadas en las simulaciones han sido extraídas de la biblioteca SNDlib [49].

Figura 4.1: Consumo total de energía en una red de 70 nodos

La figura 4.1 muestra el consumo total de energía en una red con 70 nodos para distintos números de demandas. La figura compara el consumo de potencia total resultante de aplicar el algoritmo heurístico propuesto, el consumo total si todos los switches de la topología estuvieran activos (pero sólo los enlaces necesarios) y el consumo total si estuvieran activos todos los switches y todos los enlaces de la topología de red. Se puede comprobar cómo el consumo resultante de la aplicación del heurístico propuesto es el óptimo. El porcentaje de ahorro está comprendido entre el 25 % y el 50 %. Por otro lado, podemos comprobar que cualquier incremento del número de demandas a la red supondrá un incremento en el consumo de energía en dicha red.

Figura 4.2: Consumo total de energía en una red de 20 nodos

La figura 4.2 muestra el consumo total de potencia en una red con 20 nodos también en función de distintos números de demandas. En dicha figura se confirman los mismos patrones que ya vimos en la figura 4.1 y se corrobora que el algoritmo heurístico propuesto ofrece un mejor rendimiento energético. A partir de la figura 4.3 podemos observar que cuando el número de demandas de tráfico es pequeño, el porcentaje de ahorro de consumo de potencia en el escenario de $N = 20$ nodos es similar al del escenario $N = 70$. Por el contrario, a medida que el número de demandas de tráfico va aumentando, el porcentaje de consumo de potencia ahorrada es inferior en el caso de la topología de $N = 20$. Esto se debe a que, en la topología de 20 nodos, cualquier incremento de la demanda de tráfico supondrá que la mayoría de los nodos de la red deban entrar en funcionamiento para poder satisfacer a la totalidad del tráfico demandado. Por el contrario, en el caso de $N = 70$, al contar con un mayor número de nodos disponibles, podremos encaminar las nuevas demandas de tráfico entre un mayor número de rutas disponibles, con lo que se consigue un mayor porcentaje de ahorro del consumo energético.

4.3. Optimización del consumo de energía en redes SDN

Figura 4.3: Porcentaje de energía ahorrada

Figura 4.4: Incremento de la potencia total cuando se activan nuevas demandas

La figura 4.4 muestra los incrementos de la potencia total de una red a medida que se incrementa el número de demandas. Puede observarse que el incremento de la energía total alcanza los valores más elevados cuando se da respuesta a las primeras demandas. Esto es así porque habrá que activar los nodos y los enlaces necesarios para permitir la conectividad entre los diferentes orígenes y destinos. A medida que la demanda aumenta, el incremento no es tan grande, puesto que las nuevas rutas se establecerán a través de nodos ya activados empleando enlaces ya activados. Sólo en el caso de que ésto no sea posible, se activarán nuevos switches o enlaces, lo que se representa en la figura con picos de consumo de energía.

4.4. Conclusiones

Como solución al problema de optimización energética planteado, se ha propuesto un método de optimización basado en los factores que influyen al aumento de consumo en los switches OpenFlow. Motivado por la excesiva complejidad de los modelos ILP, se ha desarrollado un algoritmo heurístico con el fin de resolver el problema de minimización del consumo de energía. Dicho modelo reduce drásticamente los tiempos de cálculo, ofreciendo una solución ágil y eficiente, aplicable a redes de gran tamaño.

A partir de una serie de simulaciones experimentales, usando topologías reales y para diferentes demandas de tráfico, hemos podido demostrar que nuestra propuesta permite alcanzar un ahorro de consumo energético de un 40 % o incluso más, mejorando con ello otros prototipos comentados en este capítulo.

Encaminamiento no IP basado en claves en redes SDN OpenFlow

5.1. Introducción

En los últimos años ha surgido la necesidad de comunicación de host a host/contenido/servicio, más allá de la tradicional comunicación host-to-host ofrecida por el protocolo IP. Hoy en día, obtener este tipo de comunicación directamente a nivel de red, es cada vez más demandado, en el marco de los nuevos escenarios de red, como el Internet de las cosas y los escenarios data-centers.

Apoyándonos en las soluciones de encaminamiento basadas en clave (*Key-Based Routing*, KBR) [50] que junto a las tablas Hash distribuidas (*Distributed Hash Tables*, DHT [51]) han ofrecido soluciones de compartición de contenidos en redes overlay mantenidas sobre Internet, en este capítulo vamos a proponer una solución de encaminamiento no IP denominada OFC-KBR (*OpenFlow Compatible Key-Based Routing*).

La propuesta OFC-KBR ofrece una solución de enrutamiento en la que los terminales finales son identificados mediante identificadores virtuales (claves) y no por direcciones IP. En nuestra implementación, el identificador virtual se obtiene del nombre del host. El identificador del host puede definirse en función de las necesidades del servicio que va a utilizar la solución OFC-KBR.

A diferencia de las soluciones tradicionales de enrutamiento basadas en claves que se definen a nivel de aplicación, la novedad de nuestra propuesta es el hecho de que se implementará completamente a nivel de red, sin necesidad de ninguna infraestructura adicional. Esto es posible gracias al uso de la arquitectura SDN, que mantiene una visión global de la red y además ofrece un amplio control de sus dispositivos. Utilizando el protocolo OpenFlow, los elementos de red quedarán configurados para poder encaminar los paquetes adecuadamente. Un controlador SDN (o un conjunto distribuido de controladores) obtiene e inserta en la *flow table* OpenFlow el conjunto de entradas requerido para ofrecer este servicio de enrutamiento basado en claves. El hecho de que la solución pro-

puesta no requiere ninguna modificación del estándar OpenFlow ni de los elementos de red basados en OpenFlow, permite que OFC-KBR coexista con los protocolos L2/L3 tradicionales. Por ejemplo, las entradas de las *flow table* asociadas con la solución OFC-KBR pueden coexistir con otras entradas que permitan a los elementos de la red el enrutamiento basado en direcciones IP o direcciones MAC.

El trabajo presentado en este capítulo ha dado lugar a la siguiente publicación:

- A. F. de la Cruz, P. Manzanares-Lopez, J. P. Muñoz-Gea and J. Malgosa-Sanahuja. “OpenFlow Compatible Key-Based Routing Protocol: Adapting SDN Networks to Content/Service-Centric Paradigm”. En: Journal of Network and Systems Management, vol. 27, num. 3, Jul. de 2019, págs. 730-755.

5.2. Descripción de la solución de encaminamiento OFC-KBR

5.2.1. Introducción

Como indicamos anteriormente, en este capítulo presentamos OFC-KBR [52], una solución de enrutamiento no IP basada en el uso de claves e implementada directamente en la capa de red usando el potencial del protocolo OpenFlow en las redes SDN. La propuesta OFC-KBR se basa en la definición de identificadores virtuales (vids), que permiten identificar a todos los elementos del sistema. En el sistema propuesto, los identificadores virtuales se emplearán para referirnos tanto a los hosts y elementos de red, como a los contenidos y servicios.

La propuesta más parecida a nuestro estudio es VIRO [53]. La idea principal de VIRO es la definición de un espacio de identificadores virtuales estructurado y dependiente de la topología sobre el que se mapearán tanto las direcciones físicas de los dispositivos (por ejemplo, las direcciones Ethernet) como las direcciones de mayor nivel (por ejemplo las direcciones IPv4/IPv6). Teniendo en cuenta las características del espacio de identificadores virtuales, VIRO define un algoritmo KBR para construir las tablas de encaminamiento de los dispositivos, así como los procedimientos de búsqueda de objetos y el reenvío de paquetes.

Si bien tanto VIRO como OFC-KBR son soluciones basadas en mecanismos KBR, la principal diferencia entre ambas es la implementación de cada una de ellas. A diferencia de VIRO, OFC-KBR no requiere ningún tipo de infraestructura adicional ni requiere ninguna modificación de los elementos de red que implementan OpenFlow.

Por ello, y a diferencia de VIRO, la solución propuesta OFC-KBR es totalmente compatible con el estándar de OpenFlow, no necesitando ninguna modificación ni extensión, ni ningún cambio en los dispositivos OpenFlow (por ejemplo, los OpenvSwitches).

5.3. Tecnologías existentes empleadas en OFC-KBR

Según el estándar OpenFlow, los valores de los campos empleados en la estructura de *matching* de una entrada de la *flow table* pueden corresponderse con un valor concreto, con *wildcards* (es decir, comodines que indican cualquier valor) o con un valor asociado a una máscara. Como se describirá en detalle en los siguientes apartados, nuestra propuesta se basa en la obtención de unos determinados intervalos numéricos que se emplearán para configurar adecuadamente las tablas de enrutamiento de los elementos de la red. Sin embargo, no siempre es posible representar cualquier intervalo numérico mediante una única entrada que contenga valores permitidos por el estándar OpenFlow.

Para resolver este problema, nuestro trabajo utilizará una tecnología de representación de rangos conocida como de *expansión de prefijos* [54]. El objetivo de este tipo de representación de rangos es convertir cada rango en un conjunto de subrangos que podrán ser incluidos en una *flow table* mediante entradas compatibles con el estándar OpenFlow. Emplear este tipo de representación supone un coste en cuanto a número de entradas, puesto que un rango puede requerir ser convertido en un grupo de subrangos.

En términos de entradas en las tablas de flujo OpenFlow, Orange [55] propone una solución más eficiente para implementar la búsqueda de rangos en los switches para OpenFlow. Esta solución es una adaptación de PIDR (*Point Intersection Disjoint Ranges*) [56]. PIDR es un algoritmo para TCAMs (*Ternary Content-Addressable Memories*) que puede ser fácilmente implementado en hardware para resolver el problema de búsqueda de rangos. La mayor parte de la funcionalidad de Orange se puede implementar utilizando tablas OpenFlow, pero también depende del uso de una tabla de comparación. Éste es el principal inconveniente de Orange, ya que esta tabla de comparación no se puede implementar utilizando una tabla OpenFlow.

De hecho, lo que proponemos en este trabajo es una forma de implementar tablas de comparación usando solo tablas OpenFlow.

5.3. Tecnologías existentes empleadas en OFC-KBR

5.3.1. Mecanismo de búsqueda Chord

El principio básico de los protocolos DHT es la asociación entre nodos y objetos, y la construcción de estructuras de enrutamiento distribuidas para localizar los objetos eficientemente. En Chord, los nodos y los identificadores de objetos están situados en un círculo formado por 2^m valores ($id \in [0, 2^m - 1]$). Cada nodo es responsable de un conjunto de claves. En particular, una clave k es asignada al nodo cuyo identificador es igual a k , o, si no existe, se asigna al primer nodo existente en el sentido de las agujas del reloj. Ese nodo se llama sucesor. Cada nodo n mantiene su predecesor (el primer nodo en sentido contrario a las agujas del reloj), su sucesor (el primer nodo en sentido de las agujas del reloj) y una lista de nodos adicionales llamada *finger table*. La *finger table* está compuesta por m entradas, donde la entrada i -th de la tabla asocia la clave $n + 2^i$ con la clave del sucesor. Cuando el nodo n busca una clave k , comprueba si k se encuentra entre

él y su sucesor. Si es así, el sucesor es el nodo buscado. De lo contrario, comprueba su *finger table* de abajo hacia arriba para localizar al predecesor inmediato de k . Entonces, el nodo n requiere que ese nodo se encargue de la búsqueda. Este procedimiento se realiza recursivamente por diferentes nodos hasta que se encuentra el nodo responsable de la clave k . El número de nodos con los que se debe contactar para encontrar un sucesor de una determinada clave es el siguiente $O(\log_2(N))$, donde N es el actual número de nodos activos. La figura 5.1 muestra un ejemplo de búsqueda en un sistema Chord, donde $m = 6$ bits.

Figura 5.1: Un ejemplo de búsqueda en un sistema Chord, donde $m = 6$ bits. El nodo 8 está buscando un objeto cuyo identificador clave es el 53

5.3.2. Representación de rangos basada en expansión de prefijos

La especificación de OpenFlow 1.3 [57] define la longitud y los posibles valores de los campos que pueden emplearse en la estructura de *matching* de una entrada de la tabla OpenFlow como se observa en la tabla 5.1.

Algunos de estos campos podrán tomar únicamente valores concretos. Otros pueden contener un rango de valores representados mediante el uso de una máscara. Sin embargo, OpenFlow no define cómo pueden ser incluidos rangos de valores que no pueden ser representados mediante el uso de una máscara de bits. Una forma de resolver este problema es la utilización de la técnica de representación de rangos conocida como *expansión de prefijos* [54].

5.3. Tecnologías existentes empleadas en OFC-KBR

Field	Bits	Description
OXM_OF_IN_PORT	32	Ingress port
OXM_OF_ETH_DST	48	Ethernet destination port. Can arbitrary mask
OXM_OF_ETH_SRC	48	Ethernet source port. Can arbitrary mask
OXM_OF_ETH_TYPE	16	Ethernet type pf the OpenFlow packet payload, after VLAN tags
OXM_OF_IPV4_PROTO	8	IPv4 or IPv6 protocol number
OXM_OF_IPV4_DST	32	IPv4 destination address. Can be subnet mask or arbitrary bitmask
OXM_OF_IPV4_SRC	32	IPv4 source address. Can be subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	128	IPv6 destination address. Can be subnet mask or arbitrary bitmask
OXM_OF_IPV4_SRC	128	IPv6 source address. Can be subnet mask or arbitrary bitmask
OXM_OF_TCP_SRC	16	TCP source port
OXM_OF_TCP_DST	16	TCP source port
OXM_OF_TCP_SRC	16	UDP source port
OXM_OF_TCP_DST	16	UDP source port

Tabla 5.1: Match fields de la especificación OpenFlow 1.3

La idea principal de esta técnica consiste en la conversión de un rango que no puede ser representado mediante la utilización de una máscara de bits, en un conjunto de subrangos que sí pueda, y, por lo tanto, poder incluirse en una *flow table* mediante una entrada por subrango. El algoritmo utilizado para obtener los subrangos a partir un determinado rango se describe en la tabla 5.2.

Dicho algoritmo consiste en obtener el Prefijo Común Más Largo (*Longest Common Prefix*, LCP) de los valores extremos del rango, y los prefijos extendidos (*Extended LCP*, ELCPs) 0-ELCP y 1-ELCP. Los ELCPs se calculan a partir del LCP añadiendo un bit más. Para un LCP P el 0-ELPC es P0 y el 1-ELPC es P1. A continuación, utilizando los correspondientes valores 0-ELCP, 1-ELCP y los valores de inicio y final, el rango original se divide en dos subrangos. Para cada subrango, el proceso se repite hasta que se obtiene un conjunto de subrangos que pueden ser representados mediante la utilización de una máscara de bits.

Como ejemplo, el rango $[32, 54] = [00100000, 00110110]$ puede ser expresado por el siguiente conjunto de rangos enmascarados:

- $[32, 47] = [0010 * * * *]$,
- $[48, 51] = [001100 * *]$,
- $[52, 53] = [0011010*]$,
- y $[54] = [00110110]$.

por lo que el rango original que no podría ser incluido en una única entrada de la tabla OpenFlow, pero podría incluirse mediante la inserción de cuatro entradas.

```

1: Input:  $[s, t] = [s_0 s_1 s_2 \dots s_{k-1}, t_0 t_1 t_2 \dots t_{k-1}]$ 
2: Output: a set of valid range
3: function bitmasked_ranges (s,t):
4:   obtain the LCP =  $p_0 p_1 \dots p_{p-1}$ 
5:   using 0-ELPC, determine the subrange  $[p_0 p_1 \dots p_{p-1} 0 s_{p+1} \dots s_{k-1}, p_0 p_1 \dots p_{p-1} 0 1 \dots 1]$ 
6:   if the subrange can be bitmasked:
7:     valid subrange identified
8:   else:
9:     new subrange  $[s', t'] = [p_0 p_1 \dots p_{p-1} 0 s_{p+1} \dots s_{k-1}, p_0 p_1 \dots p_{p-1} 0 1 \dots 1]$ 
10:    bitmasked_ranges (s,t)
11:   using 1-ELPC, determine the subrange  $[p_0 p_1 \dots p_{p-1} 1 0 \dots 0, p_0 p_1 \dots p_{p-1} 1 t_{p+1} \dots t_{k-1}]$ 
12:   if the subrange can be bitmasked:
13:     valid subrange identified
14:   else:
15:     new subrange  $[s', t'] = [p_0 p_1 \dots p_{p-1} 1 0 \dots 0, p_0 p_1 \dots p_{p-1} 1 t_{p+1} \dots t_{k-1}]$ 
16:     bitmasked_ranges (s', t')

```

Tabla 5.2: Algoritmo para obtener los rangos representables mediante un valor y una máscara de bits utilizando la expansión de prefijos

5.4. OFC-KBR

5.4.1. Asignación de identificadores virtuales

En OFC-KBR, los identificadores virtuales (*vids*) tanto de los elementos de red como de los terminales finales (*endpoints*) se definirán como cadenas de L bits donde se distinguen dos partes de $L/2$ bits cada una:

- en el caso de un elemento de red, los primeros $L/2$ bits serán el resultado de aplicar una función hash al identificador OpenFlow del dispositivo (*datapath identifier*), mientras que los últimos $L/2$ bits se fijarán a 1,
- para los terminales finales, los primeros $L/2$ bits serán el resultado de aplicar la función hash al identificador Openflow (*datapath identifier*) del elemento de red al que está conectado el terminal, mientras que los últimos $L/2$ bits serán el resultado de aplicar la función hash a su propio nombre. La definición de este nombre no será fija, ya que podrá ser un texto plano o incluso jerárquico, dependiendo de las necesidades del servicio que vaya a emplear el mecanismo de encaminamiento OFC-KBR propuesto.

En la implementación de nuestra propuesta OFC-KBR hemos utilizado el campo de direcciones IPv4 para almacenar los identificadores de red. Por ello, puesto que una dirección IPv4 es de 32 bits, 16 de ellos se emplean para identificar los elementos de red y los restantes 16 estarán a 1. Estos 16 bits nos permitirán disponer de hasta 2^{16} dispositivos de red en nuestra topología. Si fuera necesario trabajar en redes más grandes, podríamos también utilizar el campo de direcciones IPv6, de 128 bits.

5.4. OFC-KBR

En la figura 5.2 se muestra un ejemplo de topología física de una red SDN, mostrándose un posible caso de asignación de identificadores virtuales con una longitud de 10 bits. Como ejemplo, observamos que el host 1 (*h1*) tiene el identificador virtual *1010100110*. En dicha asignación, los primeros 5 bits corresponderían al valor hash del identificador del dispositivo Openflow *dp3*, mientras que los últimos 5 bits corresponderían al valor hash del nombre textual asociado a *h1*. De igual modo, decir que el identificador virtual del dispositivo *dp3* es *1010111111*, en el que los primeros 5 bits corresponderían al valor hash del identificador Openflow del dispositivo, mientras que los últimos 5 bits se ajustarán a *1*.

Figura 5.2: Ejemplo de topología física de una red SDN y la correspondiente asignación de *vids*, donde los *vids* tienen una longitud de 10 bits

5.4.2. Información de encaminamiento basada en el uso de claves

OFC-KBR se inspira en el mecanismo de búsqueda de las soluciones DHT. Aunque podría considerarse cualquiera de las posibles soluciones DHT existentes, hemos elegido para la implementación de nuestra propuesta el protocolo Chord [58] debido tanto a su simplicidad como a su uso generalizado en la investigación. Cada nodo Chord conoce y mantiene la siguiente información, necesaria para realizar el proceso de búsqueda: una tabla denominada *finger table* (formada por L entradas), la identidad del nodo sucesor (*sucesor*) y la identidad del nodo predecesor (*predecessor*).

En nuestro trabajo hemos tomado como referencia el algoritmo de búsqueda Chord para implementar una solución de enrutamiento, sin embargo, no implementamos la *finger table* ni el procedimiento de búsqueda Chord directamente.

Un requisito de diseño de nuestra propuesta exige que OFC-KBR sea totalmente compatible con la arquitectura actual de los switches OpenFlow, a fin de que cualquiera de los elementos de red pueda utilizar única y exclusivamente las tablas OpenFlow para tomar las decisiones de enrutamiento que precise. Como consecuencia de ello, será necesario

traducir el contenido de la *finger table* a una serie de entradas que puedan ser almacenadas en la *flow table* del switch OpenFlow. Además habrá que convertir el procedimiento de búsqueda de Chord en un procedimiento compatible con la búsqueda de entradas adecuadas (*matching*) empleado por OpenFlow. El campo empleado para hacer el *matching* en la tabla Openflow determina el rango de los identificadores virtuales y el campo de acción indica qué hacer con el paquete incluido en ese rango de valores.

El controlador SDN utilizará el conocimiento global de la red para realizar esta traducción y configurar la *flow table* de cada elemento de manera adecuada. La tabla 5.3 muestra el contenido de la *finger table* de un nodo cuyo identificador virtual es n . La tabla 5.4 muestra el conjunto de rangos obtenidos a partir de la *finger table* del nodo n . En la tabla 5.4, $ft(n + 2^i)$ se refiere al valor de la entrada $n + 2^i$ de la *finger table*, es decir, el id del elemento de red responsable de valor $n + 2^i$. Además, $\rightarrow x$ equivale a la acción 'enviar al switch x '.

Finger Table of node n	
$n+2^0$	node resp. of $n+2^0$
$n+2^1$	node resp. of $n+2^1$
$n+2^2$	node resp. of $n+2^2$
....	
$n+2^{L-2}$	node resp. of $n+2^{L-2}$
$n+2^{L-1}$	node resp. of $n+2^{L-1}$
predecessor	

Tabla 5.3: *Finger Table* de un nodo cuyo identificador virtual es n

Equivalent ranges and forwarding actions for node n	
$]predecessor, n]$	$\rightarrow n$
$[n+2^{L-1}, predecessor]$	$\rightarrow ft(n+2^{L-1})$
$[n+2^{L-2}, n+2^{L-1} [$	$\rightarrow ft(n+2^{L-2})$
$[n+2^{L-3}, n+2^{L-2} [$	$\rightarrow ft(n+2^{L-3})$
....	
$[n+2^0, n+2^1 [$	$\rightarrow ft(n+2^0)$

Tabla 5.4: Rangos y acciones asociadas correspondientes a un nodo de identificador virtual n . $ft(n + 2^i)$ hace referencia al valor correspondiente a la entrada $n + 2^i$, indicando el switch responsable del identificador $n + 2^i$

A modo de ejemplo, la figura 5.3 muestra la topología virtual asociada a la topología física de la figura 5.2, donde se ha asignado a los terminales y dispositivos de red su

5.4. OFC-KBR

correspondiente identificador virtual según lo definido anteriormente. Además se muestra el contenido de la *finger table* de dos dispositivos (*dp2* y *dp6*) y el conjunto de rangos equivalentes en sus tablas OpenFlow.

Puede observarse que, dependiendo de los valores de los rangos obtenidos y de las acciones asociadas, algunos de los rangos pueden ser unificados. En este ejemplo, tanto para *dp2* como para *dp6*, los once rangos iniciales pueden ser reducidos únicamente a 4 rangos.

Figura 5.3: Teniendo en cuenta la topología física mostrada en la figura 5.2, esta figura muestra la topología virtual obtenida con OFC-KBR. Las tablas de la parte inferior muestran un ejemplo de los rangos obtenidos correspondientes a los switches *dp2* y *dp6*

Una vez obtenidos los rangos equivalentes y las acciones asociadas, el controlador incluirá en la *flow table* de cada elemento de red las entradas necesarias. La implementación actual de OFC-KBR utiliza los campos de dirección IPv4 para incluir los valores de vid. Por consiguiente, el campo IPv4 de destino será el campo empleado en la estructura de *matching* de las entradas de la tabla OpenFlow para identificar las acciones de encaminamiento.

Debido al hecho de que algunos rangos obtenidos no podrán ser expresados por un único valor con máscara de bits, se utilizará el algoritmo de expansión del prefijo visto en el anterior apartado 5.3.2. Este algoritmo ofrece una solución totalmente compatible con las actuales arquitecturas de switch OpenFlow, a fin de poder expresar cualquier rango obtenido como un conjunto limitado de subrangos, a los cuales les corresponderá una entrada distinta en la tabla OpenFlow.

Como puede deducirse de la tabla 5.3, el número mínimo de rangos obtenidos para un determinado elemento de red será 2, lo que ocurrirá cuando todas las entradas de la *finger table* apunten al sucesor del nodo. Por el contrario, el número máximo de rangos para un elemento de la red será $L + 1$, que se dará cuando cada entrada de la *finger table* apunte a un nodo diferente. Hemos desarrollado un modelo matemático que permite conocer la probabilidad de que el número de rangos obtenidos para un determinado switch sea k en función del número total de elementos constituyentes de la red. El desarrollo de este modelo matemático permite verificar la viabilidad del sistema propuesto, comprobando que el número de entradas necesarias se mantiene en valores aceptables. Este modelo matemático ha sido validado mediante simulación (Ver Apéndice A).

5.4.3. Diferencia entre rutas físicas y rutas virtuales

La ubicación de los elementos de red dentro de la topología virtual dependerá de sus identificadores virtuales y no estará vinculada en modo alguno a la topología física. Por lo tanto, las rutas virtuales no tendrán por qué ser idénticas a las rutas físicas. Volviendo a la figura 5.3, el tercer rango obtenido para el switch *dp6* es $[543, 799]$. En este caso, la acción asociada a cualquier valor del rango $[543, 799]$ es enviar a 703 (es decir, el *dp3*). Ésto deberá traducirse en una acción compatible con el estándar OpenFlow, en concreto la acción deberá indicar cuál es el puerto a través del que *dp6* se conecta con *dp3*. Sin embargo, no hay una conexión física directa entre *dp6* y *dp3*.

Una solución simple podría ser enviar el paquete a través del puerto de salida determinado por la ruta física óptima hacia el destino. Debido a que el controlador tiene un conocimiento completo de la topología de la red, puede obtener la ruta física óptima aplicando un algoritmo de enrutamiento adecuado, como por ejemplo Dijkstra [48]. En el ejemplo anterior, la acción asociada al rango sería enviar a *dp7*, ya que la ruta física más corta hacia *dp6* sería a través de ese switch. Sin embargo, esta solución no es siempre válida, debido a la aparición de bucles, como vemos a continuación. Después de recibir el paquete, *dp7* utilizará su propia *flow table* para reenviarlo hacia el destino deseado. Si la entrada de flujo correspondiente en *dp7* determina que el paquete debe enviarse a un elemento de red cuya ruta física más corta pasa por *dp6*, se crearía un bucle.

Figura 5.4: Implementación de rutas físicas utilizando tecnología MPLS

5.5. Unión de terminales al sistema y definición del mecanismo de resolución de nombres

Para resolver este problema proponemos la creación de un túnel para la ruta física correspondiente al salto virtual entre los dos dispositivos de red, utilizando para ello la tecnología MPLS (*Multiprotocol Label Switching*), como se describe en la figura 5.4.

Mediante esta solución, cada vez que el controlador SDN inserte las entradas de flujo correspondientes a un rango determinado en un switch de la red, también insertará las entradas necesarias para implementar el túnel en todos los switches que forman la ruta física entre el switch origen y el switch destino. En el switch origen, las acciones de cada una de las entradas insertadas asociadas a un cierto rango serán la inserción de una etiqueta MPLS en el paquete y el reenvío del paquete a través del puerto adecuado. En cada switch intermedio, excepto en el último antes del destino, el controlador instalará únicamente una entrada de flujo en la que los campos de *matching* son el tipo de paquete y el valor de la etiqueta MPLS, y la acción asociada consistirá en el reenvío de dicho paquete a través del puerto de salida adecuado. Finalmente, en el último switch antes del destino, además de los campos de *matching* añadidos, el controlador añadirá la acción de eliminar la etiqueta MPLS.

5.5. Unión de terminales al sistema y definición del mecanismo de resolución de nombres

El protocolo OFC-KBR descrito en la sección anterior define el modo en que los elementos de red serán capaces de dirigir el tráfico hacia el elemento de red responsable de un cierto identificador virtual. Sin embargo, el objetivo de nuestro trabajo es ofrecer una comunicación centrada en el host/servicio/contenido.

Para conseguirlo, es necesario definir un procedimiento de unión entre los elementos finales (host, servicios o contenidos al sistema) y definir un servicio de resolución de nombres. Ambos mecanismos tienen en cuenta la definición de los identificadores virtuales empleados por OFC-KBR y los principios de la solución DHT empleada.

5.5.1. Definición de los paquetes JOIN

La unión de los puntos finales a la red se resolverá mediante la definición de un tipo de paquete denominado paquete JOIN. En la implementación del sistema basado en OFC-KBR, un paquete JOIN será un paquete en el cual:

- la clave de origen será el identificador virtual del elemento que desea unirse al sistema,
- y la clave de destino, será el identificador virtual del switch al que está conectado el elemento que desea unirse al sistema

Todos los elementos de red contendrán en su tabla OpenFlow una entrada que les permitirá identificar los paquetes JOIN, basándose en la clave de origen y la clave de destino. La acción asociada a esta entrada será el envío del paquete JOIN al controlador mediante

un mensaje OpenFlow Packet_In.

Una vez que el controlador recibe y procesa el mensaje Packet_In, instalará en los dispositivos de red que sean necesarios las entradas en las tablas OpenFlow para implementar el servicio de resolución de nombres que se describe a continuación.

5.5.2. Mecanismo de resolución de nombres

Una característica de nuestro sistema es el hecho de que cualquier terminal del sistema sólo conoce el nombre del elemento final con el que se quiere comunicar, y no su identificador virtual, que depende de su ubicación física. Esta característica será útil para adaptar nuestra solución de enrutamiento a escenarios donde los hosts sean dispositivos móviles, ya que el nombre de los puntos finales no dependerá de su ubicación física. Por lo tanto, se requerirá un mecanismo que se ocupe de la resolución del nombre de puntos finales en sus identificadores virtuales (vid).

Definimos un mecanismo de resolución de nombres distribuido que hace uso de las entradas instaladas en la *flow table* de los switches durante el proceso de unión. Así, después de recibir un mensaje Packet_In que contenga un paquete JOIN, el controlador insertará una entrada en la *flow table* del switch que generó el Packet_In, y otra entrada en la *flow table* del switch que se encargará de resolver el nombre del terminal.

Utilizando terminología DHT, este último switch será el dispositivo de red responsable del identificador virtual formado por los últimos $L/2$ bits del terminal que se une al sistema y el resto de bits puestos a 0. Puesto que en la implementación de OFC-KBR empleamos Chord, el dispositivo de red responsable de esta clave será el elemento de red cuyo identificador virtual es el siguiente valor existente, según el sentido horario, en el espacio del identificador virtual.

Por su parte, la entrada de la *flow table* instalada en el switch que generó el Packet_In permitirá el envío de cualquier paquete que vaya dirigido al terminal en cuestión (es decir, el extremo que generó el paquete JOIN de unión). Esta entrada indica el envío del paquete a través del puerto de salida adecuado. Por otro lado, la entrada de la *flow table* instalada en el otro switch implementará la resolución del nombre y actuará como un nodo intermedio en el proceso de transmisión hacia el destino.

5.5.3. Ejemplo de unión y de resolución de nombre

Esta sección describe mediante un ejemplo los procesos descritos anteriormente. Para ello, tomaremos como referencia la topología de red mostrada en la figura 5.3 y el cronograma de instrucciones detallado de la figura 5.5, que describe la unión de los hosts *h1* y *h3*, además del proceso de resolución de nombres asociado al envío de un mensaje de *h1* a *h3* y desde *h3* a *h1*.

5.5. Unión de terminales al sistema y definición del mecanismo de resolución de nombres

Figura 5.5: Cronograma que muestra la unión de *h1* y *h3*, y la resolución de nombre involucrada en el envío de mensajes de *h1* a *h3* y de *h3* a *h1*

La unión del host *h1* se describe a continuación:

1. El host *h1* (*vid* = 1010100110) está conectado al switch *dp3* (*vid* = 101010000), por lo que ambos valores se utilizarán para crear el correspondiente paquete JOIN, cuyos valores origen y destino son 1010100110 y 101011111, respectivamente.
2. empleando la primera entrada de flujo almacenada en el switch *dp3*, cuyos campos de *matching* son *src_ip4* = 10101 * * * * * y *dst_ip4* = 10101111111, el paquete JOIN es enviado al controlador mediante un mensaje Packet_In,
3. tras procesar el paquete JOIN, el controlador instala una nueva entrada en la *flow table* de *dp3*. Debido a esta entrada, cualquier paquete recibido por *dp3* y destinado a *h1* será reenviado a *h1* a través del puerto adecuado (en este caso, sería *p_{out}*),
4. puesto que los paquetes enviados por cualquier terminal al terminal *h1* tendrán como clave destino 0011000000 (es decir, el nombre del host y el resto a 0s), para implementar el procedimiento de resolución de nombres, el controlador instalará una entrada en el switch responsable de esta clave, en este caso *dp6*,
5. la entrada instalada en *dp6* permitirá la resolución de nombre mediante la modificación

de la clave destino del paquete recibido. En este ejemplo, se asignará como nuevo valor la clave *1010100110*, es decir, el identificador virtual de *h1*,

6. por último, *dp6* reenviará usando sus entradas de enrutamiento OFC-KBR.

A continuación, la línea de tiempo describe el envío de un paquete de *h1* a *h3* (y de *h3* a *h1*). En el primer caso, la clave de destino del mensaje es *1010000000*, clave que se obtiene del nombre de *h3*. Utilizando las entradas de enrutamiento OFC-KBR, el paquete llega al switch responsable de la resolución del nombre, en este caso, al propio switch. A continuación, el paquete se envía utilizando las entradas de enrutamiento OFC-KBR hacia *h3*. En el segundo caso, la clave de destino del mensaje es *0011000000*, que es la clave que se obtiene del nombre de *h1*. Utilizando las entradas de enrutamiento OFC-KBR, el mensaje llega al switch responsable de la resolución del nombre, en este caso, *dp6*. Finalmente el mensaje es enviado utilizando las entradas de enrutamiento OFC-KBR hacia *h1*.

5.6. Evaluación del sistema propuesto

Esta sección evaluará el rendimiento del sistema basado en el protocolo OFC-KBR propuesto. Hemos realizado nuestras simulaciones utilizando Mininet. El controlador SDN empleado es RYU [21]. Las simulaciones se han realizado considerando tres ejemplos de topologías de redes existentes extraídas de Internet Topology Zoo [59], que se muestran en las figuras 5.6 (a), 5.6 (b) y 5.6 (c) respectivamente.

Figura 5.6: Topologías. SwitchL3 (39 nodos y 62 enlaces), Dfn (56 nodos y 87 enlaces) and GtsCe (148 nodos y 192 enlaces)

Para cada topología de red se han realizado diez simulaciones. Por definición, el identificador virtual de cada nodo se obtiene a partir de su identificador OpenFlow (*datapath identifier*) mediante la aplicación de una función hash. Cada simulación se ha llevado a cabo asignando un conjunto diferente de identificadores OpenFlow a los nodos.

5.6. Evaluación del sistema propuesto

5.6.1. Conversión de las finger tables en rangos

OFC-KBR se basa en la conversión de las *finger tables* del procedimiento de búsqueda del Chord en intervalos que se utilizarán para rellenar las tablas OpenFlow de los nodos de la red. En esta sección vamos a evaluar esta conversión. Para cada topología de red se han realizado 10 simulaciones. La figura 5.7 (a) muestra la distribución de probabilidad del número de rangos obtenidos para cada nodo a partir de las simulaciones. La figura 5.7 (b) muestra los valores teóricos obtenidos utilizando el modelo matemático descrito en el **Apéndice A**. Como era de esperar, a medida que aumenta el número de nodos en la topología, el gráfico se desplaza hacia la derecha, dando como resultado un mayor número medio de entradas para cada nodo. Decir también que a medida que aumenta el número de nodos de cada topología, disminuirá la probabilidad de que los rangos obtenidos puedan unirse, dado que las diferentes entradas de la *finger table* apuntarán a un mayor número de nodos diferentes. En concreto, los valores medios obtenidos para cada una de las tres topologías es:

Figura 5.7: Distribución de probabilidad del número de rangos obtenidos para un nodo. $L = 32$ bits. **(a)** Para una simulación usando Mininet. **(b)** Usando el modelo matemático descrito en el Apéndice A

- 6,5743, para la topología Switch-L3 (39 nodos),
- 7,1447, para la topología DFN (56 nodos),
- 8,5351, para la topología GTSCe (148 nodos).

Estos valores medios son similares a los valores obtenidos para la longitud máxima teórica en Chord, que vendrán dados por $\log_2(N)$, donde N es el número de nodos, y que son:

- 5,2854, para la topología Switch-L3 (39 nodos),
- 5,8074, para la topología DFN (56 nodos),
- 7,2095, para la topología GTSCe (148 nodos).

5.6.2. Ocupacion de tablas OpenFlow

Como se explicó anteriormente, OFC-KBR se basa en la traducción de cada uno de los rangos obtenidos a partir de las entradas de la *finger table*, en entradas para la tabla OpenFlow que asocie a cada uno de los rangos obtenidos una acción de reenvío. Sin embargo, como se explicó en la sección 5.3.2, algunos intervalos de valores no pueden ser expresados mediante una única entrada. Para solventar este problema, OFC-KBR propone el uso del algoritmo de expansión de prefijos, de manera que cualquier rango de valores podrá ser expresado como un conjunto de rangos representables mediante la utilización de máscaras de bits.

La figura 5.8 (a) muestra el número medio de subrangos correspondientes a los intervalos obtenidos después de la conversión de las entradas de *finger table* en rangos compatibles con la inserción de entradas en las tablas OpenFlow de los dispositivos, para cada nodo de la topología Switch-L3. En el gráfico también se representa el número mínimo y máximo de rangos obtenidos en las simulaciones. Las figuras 5.9 (a) 5.10 (a) muestran los resultados obtenidos en las simulaciones considerando las topologías DFN y GTSCe, respectivamente.

(a) Número de rangos expresados mediante un valor y una máscara de bits

(b) Número de entradas en la tabla OpenFlow

Figura 5.8: Topología Switch-L3 (39 nodos y 62 enlaces)

5.6. Evaluación del sistema propuesto

(a) Número de rangos expresados mediante un valor y una máscara de bits (b) Número de entradas en la tabla OpenFlow

Figura 5.9: Topología DFN (56 nodos y 87 enlaces)

A partir del número medio de rangos obtenidos en la figura 5.7 y del número medio de subrangos enmascarados para cada nodo, definimos y calculamos el *factor de expansión* como la relación de ambos valores. Los factores de expansión obtenidos para cada una de las simulaciones son:

- 21.550, para la topología Switch-L3 (39 nodos),
- 22.005, para la topología DFN (56 nodos),
- 22.603, para la topología GtsCe (148 nodos).

Por último, vamos a evaluar la ocupación final de las tablas OpenFlow. Como se puede ver en las figuras 5.8 (b), 5.9 (b) y 5.10 (b), el número de entradas que contendrá cada switch es mayor que el número de rangos obtenidos. Este incremento se debe al hecho de transformar los caminos virtuales en caminos físicos, como se describió en la sección 5.4.3. Es destacable que el incremento entre el número de rangos y el número de entradas necesarias en las tablas de los dispositivos de red dependerá de la posición y la conectividad de los elementos de red. Dependiendo de estos dos factores, se habrán de añadir más o menos reglas MPLS a fin de poder definir el camino apropiado de enrutamiento. Por ejemplo, en la figura 5.8 (b) observamos que el switch 8 muestra un incremento más notable en relación con el número de reglas requeridas, lo que se confirma al analizar la topología Switch-L3, donde se observa que el nodo Zurich ETH tiene el mayor grado de conectividad dentro de la red. Por ello es más probable que la ruta física correspondiente a una ruta virtual atravesase este nodo, y por tanto, contenga más entradas del tipo MPLS.

El mismo comportamiento se observa en las otras dos topologías. En cualquier caso, los resultados obtenidos son aceptables, considerando que una misma tabla de enrutamiento podrá llegar a soportar de 750 a unos pocos miles de reglas. Aun así, en la sección 5.7 describimos algunas propuestas que permitirán reducir el número de solicitudes, si la tecnología OpenFlow evoluciona adecuadamente.

(a) Número de rangos expresados mediante un valor y una máscara de bits

(b) Número de entradas en la tabla OpenFlow

Figura 5.10: Topología GtsCe (148 nodos y 192 enlaces)

5.6.3. Rendimiento de la red

En esta sección vamos a evaluar el rendimiento de la red empleando para ello dos parámetros: la optimización del encaminamiento y el *throughput*.

5.6.3.1. Optimización del enrutamiento

La optimización del enrutamiento se medirá a través del parámetro *Routing Stretch*, RS , que se define como la relación entre el número de saltos físicos que utiliza OFC-KBR y el número de saltos físicos que utilizaría la ruta más corta (SP) entre dos switches i y j cualesquiera, tal como se define en la ecuación 5.1

$$RS_{ij} = \frac{L_{ij}^{OFC-KBR}}{L_{ij}^{SP}} \quad (5.1)$$

Considerando que la longitud máxima del trayecto en Chord es $\log_2(N)$, la media del RS podrá obtenerse a partir de la ecuación 5.2.

$$\overline{RS_{max}} = \frac{\log_2(N) * \overline{L^{SP}}}{\overline{L^{SP}}} = \log_2(N) \quad (5.2)$$

5.6. Evaluación del sistema propuesto

La primera fila de la tabla 5.5 muestra el valor de RS medio correspondiente a las simulaciones realizadas sobre cada una de las tres topologías, pudiendo observarse que la totalidad de los valores obtenidos es siempre superior a 1, que sería el valor ideal para el caso de una conexión directa. Como en la mayoría de las aplicaciones basadas en claves, este es el precio a pagar por usar una solución DHT. Sin embargo, la RS está limitada por $\log_2(N)$.

	Switch-L3	DFN	GTSCe
$\overline{RS}$	2.784	2.9278	4.2038
$\overline{RS_{max}}$	5.2854	5.8074	7.2095

Tabla 5.5: Routing Stretch

5.6.3.2. Throughput del enrutamiento

La estabilidad de la solución propuesta y el *throughput* obtenido dependerá de un modo directo del rendimiento de los switches. Para lograr un alto nivel de *throughput* en cualquier red, los switches deberán enrutar los paquetes entrantes lo más rápido posible. Habrá dos factores que tienen una influencia significativa en este rendimiento:

- la arquitectura de conmutación
- el retardo de búsqueda

Sin embargo, en lo que respecta a nuestra propuesta, sólo hay que tener en cuenta el retardo de búsqueda, que puede definirse como el tiempo que tarda un switch en encontrar una entrada coincidente en la *flow table*. Igual que los switches Ethernet, basados en hardware y routers IP, los switches OpenFlow aprovechan la tecnología TCAM para obtener una entrada en un solo tiempo de acceso a la memoria, que es el mínimo posible.

En un escenario de red virtualizada, el proceso de búsqueda tiene que ser realizado necesariamente por software. En este caso, OpenvSwitch será la solución más adecuada, ya que ha sido diseñado para minimizar el retardo de búsqueda [22]. Para minimizar el retardo en la búsqueda, un OpenvSwitch dispone de un módulo de kernel especializado que se define como *kernel-based datapath*. Cuando un paquete llegue a un puerto, el módulo del kernel lo procesará, extrayendo su clave de búsqueda y buscándola en la *flow table*. Si hay un flujo coincidente, ejecutará las acciones asociadas y en el caso de que no exista tal coincidencia, el paquete se pondrá en cola del *userspace* para ser procesado. En este caso, el módulo del kernel almacenará una nueva entrada para procesar paquetes recibidos a continuación del mismo tipo.

Según este modo de funcionamiento, a medida que los terminales generan tráfico y los elementos de red reenvían los paquetes, los OpenvSwitch almacenarán en el módulo del

kernel las entradas empleadas.

A modo de ejemplo, vamos a evaluar el consumo de memoria del kernel mediante el siguiente experimento. Consideraremos que hay un host conectado a cada switch. Cada host seleccionará, aleatoriamente, un destino entre el resto de hosts de la red y, tras ello, transmitirá un flujo de paquetes, a una velocidad de 8 paquetes/segundo durante 30 s. En las figuras 5.11 (a) y 5.11 (b) se muestran los resultados obtenidos en las topologías SwitchL3 y DFN, respectivamente.

Figura 5.11: Número de entradas en caché en el espacio del kernel. Como se ha comentado en la sección 5.6.2, el aumento de las entradas de los switches 49 a 52 con respecto al resto de switches de la topología DFN se debe a la posición y al alto grado de conectividad de estos switches. **(a)** Topología SwitchL3. **(b)** Topología DFN

Como conclusión y a la vista de los resultados obtenidos, podremos afirmar que aunque el número de entradas almacenadas en la tabla OpenFlow es más alto (véanse en las figuras 5.9 y 5.10), el número de entradas que se almacenan en caché será sustancialmente

5.7. Propuesta de modificación del standard OpenFlow para mejorar el rendimiento de OFC-KBR

menor.

A la vista de la figura 5.11 (b), y según se trató en la sección 5.6.2, el aumento del valor del número de entradas correspondientes a los switches 49 y 52 con respecto al resto de switches se debe a su posición y al alto grado de conectividad de dichos switches.

5.7. Propuesta de modificación del standard OpenFlow para mejorar el rendimiento de OFC-KBR

Una característica clave y distintiva de la solución OFC-KBR es el hecho de que la propuesta es totalmente compatible con la versión actual del estándar OpenFlow. Este hecho determina y condiciona algunos resultados de rendimiento como la ocupación de las tablas. En esta sección, vamos a describir cómo podría mejorarse este parámetro si el estándar OpenFlow evolucionara adecuadamente permitiendo algunas acciones adicionales simples.

El número relativamente alto de entradas en las tablas de los switches OpenFlow está motivado por la imposibilidad de expresar la mayoría de los rangos obtenidos con un valor que emplee máscara de bits. Sin embargo, este problema no existe cuando el identificador virtual del switch es 0. Como puede extraerse de la tabla 5.3, en el caso $n = 0$, los últimos $(L - 1)$ rangos obtenidos pueden expresarse siempre mediante un valor con máscara de bit. Usando esta característica, el número de entradas en las tablas de un switch OpenFlow podría reducirse siempre que, una vez el controlador obtuviese los subrangos y las acciones de reenvío para cada switch, hubiese una traducción de los rangos, restando el valor (*vid*) propio del switch a cada límite de rangos. Esta resta sería equivalente al desplazamiento del nodo hasta que el valor *vid* coincidiese con el valor inicial 0. El desplazamiento o sustracción se aplicaría a los rangos, pero nunca a las acciones.

Por tanto, si los rangos que van a utilizarse en un switch para reenviar un determinado paquete se han obtenido como se ha descrito anteriormente, cuando el switch reciba un paquete para ser enrutado hacia un destino, tendrá que restar su *vid* al valor del destino antes de encontrar el rango adecuado y así determinar las acciones a aplicar. Una vez se identifique la entrada de flujo adecuada, las acciones asociadas a dicha entrada incluirán la suma del valor del *vid* que se restó.

Para poder implementar esta propuesta que reduce significativamente el número de entradas, es necesario modificar el protocolo OpenFlow para ampliar el conjunto de acciones que pueden asociarse a una entrada de flujo. En concreto, el conjunto de nuevas acciones requeridas deberían permitir la modificación de los campos IP de origen o destino mediante la adición o sustracción de un determinado valor que debería incluirse en el mensaje OpenFlow junto a la acción como otro parámetro más.

5.8. Conclusiones

En este capítulo hemos propuesto un mecanismo de enrutamiento no basado en direcciones IP que podría coexistir con los protocolos de *L2/L3*. Entre sus principales objetivos, la solución OFC-KBR propuesta implementa en las redes SDN un paradigma que no está centrado en el host. Para alcanzar este objetivo, hemos propuesto un protocolo de enrutamiento inspirado en las soluciones KBR-DHT, ampliamente desplegadas en redes superpuestas. Una solución KBR-DHT es un sistema distribuido que almacena pares (clave, valor) y en el que cualquier nodo participante puede recuperar el valor asociado a una clave determinada.

Para adaptar este tipo de mecanismos, a fin de que puedan ser ejecutados en una implementación a nivel de red, será necesario que los switches sean capaces de tomar decisiones de enrutamiento basadas en rangos numéricos. Sin embargo, los switches OpenFlow actuales sólo serán capaces de tomar decisiones basadas en coincidencias y en enmascaramiento de bits. En este trabajo, se ha propuesto un mecanismo compatible con OpenFlow para definir rangos utilizando máscaras de bits, lo que hace que sea posible la implementación del mecanismo de enrutamiento Chord.

Hemos evaluado las prestaciones de nuestra propuesta en diferentes topologías de redes reales, demostrando que la implementación es viable. Por último, se propone una alternativa que reduciría el tamaño de las tablas de enrutamiento a costa una ampliación del conjunto de acciones disponibles por el estándar OpenFlow.

Conclusiones

El crecimiento actual de los dispositivos conectados a Internet y el incremento de la demanda de las TIC estimado para los próximos años, requieren grandes transformaciones en las infraestructuras de las redes actuales. La aparición de las redes SDN ha supuesto una nueva línea de investigación para afrontar los problemas operacionales que nos encontramos en las redes tradicionales. Con el despliegue de redes basadas en SDN, logramos una arquitectura de red dividida en el plano de control y el plano de datos. Gracias a esta división, obtenemos unas redes más ágiles, escalables, gestionables y seguras.

Para analizar el comportamiento de la red, se han venido utilizando varias técnicas de monitorización que suelen necesitar modificaciones hardware en los dispositivos de red o una instalación software. Las redes SDN, junto al protocolo OpenFlow, nos ofrecen una nueva línea de investigación con la que poder monitorizar y descubrir elementos de la red de una manera sencilla. El protocolo OpenFlow posee herramientas que nos ayudan a la monitorización de la red, pudiendo obtener estadísticas de flujo en tiempo real a través de los switches.

En términos de monitorización de red, SDN permite obtener una solución de monitorización ajustada a las necesidades específicas de la red. Mediante el uso de SDN, somos capaces de obtener una visión completa de la misma, que incluye los nodos y enlaces. Además, la solución es capaz de obtener un análisis estadístico preciso de cada flujo que transcurre por la red.

En esta tesis hemos elaborado un mecanismo para mejorar la monitorización de la red ante posibles fallos que puedan suceder (como la desconexión de un switch). Basándonos en técnicas de monitorización activas, inyectamos paquetes de prueba en la red para ser capaces de medir los parámetros de QoS, tal como el *throughput* y el *delay*.

El descubrimiento de la topología de red para su posterior monitorización, también ha sido sujeto de estudio en esta tesis. Como hemos podido ver, actualmente los mecanismos de descubrimiento de la topología suelen realizarse mediante el protocolo LLDP, aportando conocimiento sobre los nodos y enlaces. Sin embargo, ningún protocolo es capaz

de identificar los hosts existentes en la red hasta que generen tráfico. En esta tesis se ha realizado un estudio avanzado de los mensajes OpenFlow generados ante la conexión y desconexión de terminales, y se ha propuesto e implementado un mecanismo de descubrimiento de hosts en la red.

Otra de las líneas de investigación que ha sido analizada en esta tesis es la influencia de las redes SDN sobre el tan necesario ahorro energético en las redes. La reducción del consumo de energía de las redes de comunicación se establece como una de las prioridades a nivel global dado el considerable consumo de energía eléctrica de las TIC. Con las crecientes necesidades, la forma de gestión de las redes debe evolucionar no sólo en términos de aprovisionamiento de servicios, sino también en la capacidad de mejorar la eficiencia energética reduciendo las emisiones de CO_2 . Dado que la eficiencia energética de un sistema de telecomunicaciones se basa principalmente en el consumo de energía de sus componentes, una estrategia eficaz para reducir los problemas de energía es el enrutamiento inteligente y la correcta gestión de toda la configuración de la red, poniendo en modo sleep los dispositivos de red que no están siendo utilizados. En esta tesis se ha propuesto un estudio centrado en el enrutamiento *green*, logrando aumentar la eficiencia energética de las redes basadas en SDN, reduciendo el número de elementos de red activos.

La última línea de investigación abordada en esta tesis es el enrutamiento no IP adecuado para redes SDN. Durante los últimos años, las comunicaciones host-to-host /contenido/servicio han vivido un incremento de su demanda, en detrimento de las comunicaciones host-to-host ofrecidas por las soluciones de enrutamiento IP tradicionales. Este tipo de encaminamiento ha sido hasta ahora suficiente para la mayoría de aplicaciones de Internet. Sin embargo, en los últimos años, las aplicaciones desarrolladas en los nuevos escenarios de red requieren un diferente tipo de enrutamiento. Nuestra propuesta ofrece una solución de enrutamiento capaz de coexistir con los protocolos de niveles L2 y L3, en la que los end-points son identificados mediante identificadores virtuales y no mediante direcciones IP.

La propuesta realizada quiere extender las redes SDN a un paradigma de encaminamiento no limitado a direcciones IP. Para alcanzar este objetivo, hemos definido un protocolo de enrutamiento basado en soluciones KBR-DHT, las cuales han sido utilizadas en redes overlay en últimos años. Los switches OpenFlow solo son capaces de tomar decisiones basándose en *matching*. En esta tesis se ha propuesto un mecanismo de encaminamiento no IP para redes SDN OpenFlow. El sistema propuesto adapta un mecanismo de búsquedas a nivel de aplicación como es Chord a un sistema de encaminamiento a nivel de red que no requiere ningún tipo de infraestructura adicional.

Apéndice A

Supongamos una red Chord de tamaño 2^L , y formada por N nodos. El objetivo de este estudio matemático es calcular la distribución de probabilidad del número de rangos obtenidos a partir de la *finger table* de un nodo, asumiendo que el resto de los nodos $N-1$ están distribuidos aleatoriamente en el tramo: $[0, 2^L - 1]$.

Por definición, la *finger table* de un nodo estará formada por L entradas y la identidad del nodo predecesor. A partir de la información disponible en la *finger table*, se obtiene $L+1$ rangos y así como la acción de reenvío correspondiente. La entrada i asocia el rango: $[2^i, 2^{i+1}[$, con el primer nodo con valor igual o superior a 2^i , en sentido horario en el espacio virtual. Si más de una entrada consecutiva de la tabla digital apunta al mismo nodo, estos rangos se pueden combinar en un único rango. Bajo estos supuestos, vamos a obtener la probabilidad de que el número diferentes de rangos obtenidos para un nodo sea k , donde: $2 \leq k \leq L+1$, considerando que N es el número total de nodos en la red.

En lo sucesivo, el intervalo $[2^i, 2^{i+1}[$ definido a partir de la *finger table* del nodo 0 (el nodo de referencia) se denominará intervalo *i-ésimo*. Consideraremos que cualquier intervalo *i-ésimo* es un intervalo vacío si ninguno de los nodos existentes se encuentra dentro del intervalo considerado o asociado. Por otro lado, un intervalo *i-ésimo* es un intervalo no vacío si uno o más de los nodos están localizados dentro del intervalo. Por lo tanto, todos los intervalos vacíos consecutivos se fusionarán con el siguiente intervalo no vacío. Con todo ello, si existen $k-1$ intervalos no vacíos y el resto son intervalos vacíos, entonces, la *finger table* generará k diferentes rangos. Como se mostró en la figura 5.4, el rango $[predecesor, n]$, que apunta al propio nodo, siempre estará presente. Por lo tanto, habrá k diferentes rangos si la distribución de los nodos en el espacio virtual permite que el nodo 0 tenga $k-1$ intervalos no vacíos y $l-(k-1)$ intervalos vacíos. Sin embargo, debido al hecho de que cada intervalo *i-ésimo* tiene su propia capacidad (2^i), no todos los posibles casos serán válidos. En los casos no válidos, la capacidad total es inferior a $N-1$, y no es posible distribuir los $N-1$ nodos entre ellos.

A modo de ejemplo consideremos el caso en que: $l = 6$, $N = 10$ y $k = 3$. En este caso existen 15 combinaciones diferentes de 2 intervalos no vacíos y 4 intervalos vacíos, pero algunos de ellos no son válidos. Por ejemplo, si los intervalos no vacíos son los intervalos *0-ésimo* y *1-ésimo*, la capacidad sería $2^0 + 2^1 = 3$, que no será suficiente para localizar $N-1 = 9$ nodos. Sólo 9 de los 15 combinaciones tienen suficiente capacidad para localizar a los $N-1$ nodos.

Ahora, consideremos una combinación válida de intervalos del ejemplo anterior, co-

mo podría ser la (2, 3), en la que ambos intervalos dispondrían de la capacidad de localizar hasta $2^2 + 2^3 = 12$ nodos. Por lo tanto, habrá 4 posibilidades para distribuir los $N - 1 = 9$ nodos dentro de los intervalos: $n_2 = 1 ; n_3 = 8 ; n_2 = 2 ; n_3 = 7 ; n_2 = 3 ; n_3 = 6$ y $n_2 = 4 ; n_3 = 5$.

Por consiguiente, entre todas las combinaciones válidas de intervalos, debería ser considerados aquellos que cumplen con la siguiente ecuación:

$$\Gamma \in i, j, \dots, k \mid n_i + n_j + \dots + n_k = N - 1 \quad (1)$$

donde n_i es el número de nodos en el intervalo i -ésimo. Por lo tanto, la probabilidad de que la *finger table* de un nodo genere k rangos diferentes es:

$$P(k) = \sum_{\Gamma} P(n_1)P(n_2)P(n_{k-1}) \quad (2)$$

donde $P(n_i)$ es la probabilidad de que n_i del total de número de nodos $N - 1$ estén situados en el intervalo i -ésimo (con capacidad para alojar hasta nodos l_i). Finalmente, la probabilidad puede ser fácilmente obtenida usando números combinatorios como:

$$P(k) = \sum_{\Gamma} \frac{\binom{l_1}{n_1} \binom{l_2}{n_2} \dots \binom{l_{k-1}}{n_{k-1}}}{\binom{2^L - 1}{N - 1}} \quad (3)$$

El modelo matemático ha sido validado mediante una herramienta de simulación programada en lenguaje C. El simulador considerará el nodo de referencia (nodo 0) y seleccionará aleatoriamente a otros $N - 1$ nodos en un anillo virtual de $2^L - 1$ valores, comprendidos en el tramo: $[0, 2^L - 1]$. A continuación, el simulador calculará el número de rangos diferentes, asociados al nodo 0, según el procedimiento propuesto en la solución OFC-KBR, teniendo en cuenta los nodos existentes. Esta simulación se repetirá 10^6 veces para alcanzar probabilidades de estado estacionario.

La figura A.1 muestra la distribución de probabilidad del número de rangos asociados a un nodo obtenida por simulación, cuando $L = 6$ (es decir, un espacio virtual = $[0, 2^6 - 1]$) y el número de los nodos es N : 6, 12, 19, 25 y 32, que corresponderán al 10 %, 20 %, 30 %, 40 % y 50 % de la capacidad de la red. Como también se puede ver en dicha figura, a medida que aumenta el número de nodos N , la distribución de probabilidad se desplazará hacia la derecha y así mismo, a medida que aumenta el número de nodos, la probabilidad de que los rangos obtenidos directamente de las entradas de la *finger table* se puedan fusionar es menor, ya que las entradas apuntarán a nodos diferentes. En particular, en el estudio correspondiente a dicha figura, el número medio de rangos será de: 3,8; 4,8; 5,4; 5,8 y 6,2 para $N = 6, 12, 19, 25$ y 32, respectivamente.

Por otro lado, la figura A.2 muestra la distribución de probabilidad del número de rangos asociados a un nodo obtenidos por el modelo matemático. La coincidencia exacta entre ambas figuras valida claramente el modelo.

Figura A.1: Distribución de probabilidad del número de rangos obtenidos por simulación

Figura A.2: Distribución de probabilidad del número de rangos obtenidos utilizando el modelo matemático

Bibliografía

- [1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky y S. Uhlig, “Software-Defined Networking: A Comprehensive Survey”, *IEEE*, vol. 103, n.º 1, págs. 14-76, ene. de 2015.
- [2] *What is ONF*, <https://www.opennetworking.org/>.
- [3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. L. Peterson, J. Rexford, S. Shenker y J. Turner, “OpenFlow: Enabling innovation in campus networks”, *Computer Communication Review*, vol. 38, n.º 2, págs. 69-74, abr. de 2008.
- [4] T. Bakhshi, “State of the Art and Recent Research Advances in Software Defined Networking”, *Wireless Communications and Mobile Computing*, vol. 2017, págs. 1-35, ene. de 2017.
- [5] E. Haleplidis, K. Pentikousis, S. G. Denazis, J. H. Salim, D. Meyer y O. G. Koufopavlou, “Software-Defined Networking (SDN): Layers and Architecture Terminology”, *RFC*, vol. 7426, págs. 1-35, ene. de 2015.
- [6] I. Ahmad, S. Namal, M. Ylianttila y A. Gurtov, “Security in Software Defined Networks: A Survey”, *IEEE Communications Surveys Tutorials*, vol. 17, n.º 4, págs. 2317-2346, ago. de 2015.
- [7] M. Team, *Mininet: an instant virtual network on your laptop (or other PC)*, <http://mininet.org/>.
- [8] B. Lantz, B. Heller y N. McKeown, “A Network in a Laptop: Rapid Prototyping for Software-defined Networks”, en *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, oct. de 2010, págs. 1-9.
- [9] W. Stallings, “SNMP and SNMPv2: the infrastructure for network management”, *IEEE Communications Magazine*, vol. 36, n.º 3, págs. 37-43, mar. de 1998.
- [10] B. Claise, “Cisco Systems NetFlow Services Export Version 9”, *RFC*, vol. 3954, págs. 1-33, oct. de 2004.
- [11] sFlow, *Traffic Monitoring using sFlow*, <http://www.sflow.org/>.
- [12] R. Antonello, S. Fernandes, C. Kamienski, D. Sadok, J. Kelner, I. GóDor, G. Szabó y T. Westholm, “Deep Packet Inspection Tools and Techniques in Commodity Platforms: Challenges and Trends”, *J. Netw. Comput. Appl.*, vol. 35, n.º 6, págs. 1863-1878, nov. de 2012.

-
- [13] B. H. r, D. Plummer, D. Moore, Claffy y k., “Topology Discovery by Active Probing”, en *2002 Symposium on Applications and the Internet (SAINT) Workshops*, feb. de 2002, págs. 90-96.
 - [14] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely y S. C. Diot, “Packet-level traffic measurements from the Sprint IP backbone”, *IEEE Network*, vol. 17, n.º 6, págs. 6-16, nov. de 2003.
 - [15] A. Tootoonchian, M. Ghobadi e Y. Ganjali, “OpenTM: Traffic Matrix Estimator for OpenFlow Networks”, en *Passive and active measurement (PAM)*, abr. de 2010, págs. 201-210.
 - [16] J. R. Ballard, I. Rae y A. Akella, “Extensible and Scalable Network Monitoring Using OpenSAFE”, en *2010 Internet Network Management Conference on Research on Enterprise Networking*, abr. de 2010, págs. 8-8.
 - [17] M. Yu, L. Jose y R. Miao, “Software Defined Traffic Measurement with OpenSketch”, en *10th USENIX Conference on Networked Systems Design and Implementation*, ene. de 2013, págs. 29-42.
 - [18] N. L. M. van Adrichem, C. Doerr y F. A. Kuipers, “OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks”, en *2014 IEEE Network Operations and Management Symposium (NOMS)*, mayo de 2014, págs. 1-8.
 - [19] *Pox controller*, <http://github.com/noxrepo/pox>.
 - [20] A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez y J. Malgosa-Sanahuja, “Network failures support for traffic monitoring mechanisms in software-defined networks”, en *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, abr. de 2016, págs. 691-694.
 - [21] R. S. F. Community, *Build SDN agilely*, <https://osrg.github.io/ryu/index.html>.
 - [22] *Linux Foundation Collaborative Projects. Open vSwitch*, <https://www.openvswitch.org>.
 - [23] *VLC*, <https://www.videolan.org/vlc/>.
 - [24] I. F. Akyildiz, A. Lee, P. Wang, M. Luo y W. Chou, “A Roadmap for Traffic Engineering in SDN-OpenFlow Networks”, *Comput. Netw.*, vol. 71, págs. 1-30, oct. de 2014.
 - [25] P. floodlight, *Open source software for building software-defined networks*, <http://www.projectfloodlight.org/floodlight>.
 - [26] *Opendaylight*, <http://www.opendaylight.org>.
 - [27] “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)*, págs. 1-146, mar. de 2016.
 - [28] G. F. Lyon, *Nmap Network Scanning. Insecure.com LLC*, <https://nmap.org/book/toc.html>.
 - [29] S. Hong, L. Xu, H. Wang y G. Gu, “Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures”, en *2015 Network and Distributed System Security Symposium*, feb. de 2015.

BIBLIOGRAFÍA

- [30] Cisco, *Cisco Visual Networking Index: Forecast and Methodology 2017 2022*, <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>, (accessed: 01.02.2019).
- [31] S. Zeadally, S. U. Khan y N. K. Chilamkurti, "Energy-efficient networking: past, present, and future", *The Journal of Supercomputing*, vol. 62, n.º 3, págs. 1093-1118, mayo de 2011.
- [32] T. Group, *Smart 2020 - enabling the low carbon economy in the information age*, <https://www.theclimategroup.org/sites/default/files/archive/files/Smart2020Report.pdf>.
- [33] C. Lange, D. Kosiankowski, R. Weidmann y A. Gladisch, "Energy Consumption of Telecommunication Networks and Related Improvement Options", *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 17, n.º 2, págs. 285-295, mar. de 2011.
- [34] I. E. Agency, *Digitalization and Energy*, <http://www.iea.org/publications/freepublications/publication/digitalization-and-energy.html>.
- [35] Z. Movahedi, F. Dabaghi y R. Langar, "A survey on green routing protocols using sleep-scheduling in wired networks", *Journal of Network and Computer Applications*, vol. 77, n.º 4, págs. 106-122, ene. de 2017.
- [36] A. Cianfrani, V. Eramo, M. Listanti, M. Marazza y E. Vittorini, "An Energy Saving Routing Algorithm for a Green OSPF Protocol", en *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, mar. de 2010, págs. 1-5.
- [37] A. Cianfrani, V. Eramo, M. Listanti y M. Polverini, "An OSPF enhancement for energy saving in IP networks", en *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, abr. de 2011, págs. 325-330.
- [38] F. Cuomo, A. Abbagnale y S. Papagna, "ESOL: Energy saving in the Internet based on Occurrence of Links in routing paths", en *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, jun. de 2011, págs. 1-6.
- [39] F. Cuomo, A. Cianfrani, M. Polverini y D. Mangione, "Network pruning for energy saving in the Internet", *Computer Networks*, vol. 56, n.º 10, págs. 2355-2367, jul. de 2012.
- [40] F. Giroire, D. Mazauric, J. Moulierac y B. Onfroy, "Minimizing Routing Energy Consumption: From Theoretical to Practical Results", en *2010 IEEE/ACM Int'l Conference on Green Computing and Communications Int'l Conference on Cyber, Physical and Social Computing*, dic. de 2010, págs. 252-259.
- [41] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee y N. McKeown, "ElasticTree: Saving Energy in Data Center Networks", en *7th USENIX Conference on Networked Systems Design and Implementation*, ene. de 2010, págs. 249-264.
- [42] R. Bolla, R. Bruschi, F. Davoli y C. Lombardo, "Fine-Grained Energy-Efficient Consolidation in SDN Networks and Devices", *IEEE Transactions on Network and Service Management*, vol. 12, n.º 2, págs. 132-145, jun. de 2015.

- [43] R. Wang, Z. Jiang, S. Gao, W. Yang, Y. Xia y M. Zhu, “Energy-aware routing algorithms in Software-Defined Networks”, en *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, jun. de 2014, págs. 1-6.
- [44] F. Giroire, J. Moulrierac y T. K. Phan, “Optimizing rule placement in software-defined networks for energy-aware routing”, en *2014 IEEE Global Communications Conference*, dic. de 2014, págs. 2523-2529.
- [45] A. Fernandez-Fernandez, C. Cervello-Pastor y L. Ochoa-Aday, “Improved Energy-Aware Routing Algorithm in Software-Defined Networks”, en *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, nov. de 2016, págs. 196-199.
- [46] ———, “Achieving Energy Efficiency: An Energy-Aware Approach in SDN”, en *IEEE Global Communications Conference (GLOBECOM)*, dic. de 2016, págs. 1-7.
- [47] F. Kaup, S. Melnikowitsch y D. Hausheer, “Measuring and modeling the power consumption of OpenFlow switches”, en *10th International Conference on Network and Service Management (CNSM) and Workshop*, nov. de 2014, págs. 181-186.
- [48] E. Dijkstra, “A Note on Two Problems in Connexion with Graphs”, *Numb. Math.*, vol. 1, n.º 1, págs. 269-271, dic. de 1959.
- [49] S. Orlowski, R. Wessäly, M. Pióro y A. Tomaszewski, “SNDlib 1.0—Survivable Network Design Library”, *Networks*, vol. 55, n.º 3, págs. 276-286, oct. de 2010.
- [50] I. Baumgart y S. Mies, “S/Kademlia: A Practicable Approach Towards Secure Key-based Routing”, en *13th International Conference on Parallel and Distributed Systems*, dic. de 2007, págs. 1-8.
- [51] F. Dabek, M. Frans Kaashoek, D. Karger, R. Morris e I. Stoica, “Wide-area cooperative storage with CFS”, *ACM SIGOPS Operating Systems Review*, vol. 35, n.º 5, págs. 202-215, oct. de 2001.
- [52] A. Cruz, P. Manzanares-Lopez, J. P. Muñoz-Gea y J. Malgosa-Sanahuja, “Open-Flow Compatible Key-Based Routing Protocol: Adapting SDN Networks to Content /Service-Centric Paradigm”, *Journal of Network and Systems Management*, vol. 27, n.º 3, págs. 730-755, jul. de 2019.
- [53] B. Dumba, H. Mekky, S. Jain, G. Sun y Z.-L. Zhang, “A Virtual Id Routing Protocol for Future Dynamics Networks and Its Implementation Using the SDN Paradigm”, *Journal of Network and Systems Management*, vol. 24, n.º 3, págs. 578-606, jul. de 2016.
- [54] V. Srinivasan, G. Varghese, S. Suri y M. Waldvogel, “Fast and Scalable Layer Four Switching”, *SIGCOMM Comput. Commun. Rev.*, vol. 28, n.º 4, págs. 191-202, oct. de 1998.
- [55] L. Schiff, Y. Afek y A. Bremner-Barr, “Orange: multi field openflow based range classifier”, en *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, mayo de 2015, págs. 63-73.
- [56] R. Panigrahy y S. Sharma, “Sorting and searching using ternary CAMs”, *IEEE Micro*, vol. 23, n.º 1, págs. 44-53, ene. de 2003.

BIBLIOGRAFÍA

- [57] *OpenFlow switch specification: Version 1.3.5*, <https://www.opennetworking.org/software-definedstandards/specifications/>.
- [58] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek y H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for Internet applications", *IEEE/ACM Transactions on Networking*, vol. 11, n.º 1, págs. 17-32, feb. de 2003.
- [59] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden y M. Roughan, "The Internet Topology Zoo", *IEEE Journal on Selected Areas in Communications*, vol. 29, n.º 9, págs. 1765-1775, oct. de 2011.

Esta tesis ha sido apoyada por el Ministerio Español de Economía, Industria y Competitividad con la beca predoctoral FPI (BES-2014-069097). Esta investigación ha sido apoyada por la Agencia Estatal de Investigación a través del proyecto UE Project Grant TEC2016-76465-C2-1-R (AIM)