TRABAJO FIN DE GRADO

INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Implementación de Red de Área Local Extensa mediante Software Defined Networking

Autor
Bárbara Valera Muros

Directores
Juan José Ramos Muñoz
Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Julio de 2016

Implementación de Red de Área Local Extensa mediante Software Defined Networking

Autor
Bárbara Valera Muros

Directores
Juan José Ramos Muñoz
Jorge Navarro Ortiz

Departamento de Teoría de la Señal, Telemática y Comunicaciones

—

Granada, Julio de 2016

Implementación de Red de Área Local Extensa mediante Software Defined Networking

Bárbara Valera Muros

Palabras clave: Software Defined Networking (SDN), Mininet, OpenDayLight (ODL), Java, Controlador, Filtrado, Broadcast, Address Resolution Protocol (ARP), Internet Control Message Protocol version 6 (ICMPv6)

Resumen

El uso de Ethernet como tecnología de red para redes empresariales se justifica por su bajo coste y facilidad de configuración y mantenimiento. Sin embargo, estas redes, que pueden interconectar miles de dispositivos, no son muy escalables, debido en gran parte a las tormentas de *broadcasts*. Las tormentas de *broadcasts* se producen cuando varios dispositivos envían paquetes a la dirección de difusión de la red. Este fenómeno no sólo consume recursos de red, sino que que afecta al rendimiento de los dispositivos.

Las redes definidas por software, o Software Defined Networking (SDN), representan un nuevo paradigma en el que el comportamiento de los conmutadores de red se puede reprogramar de forma centralizada. Esto permite reducir costes y reconfigurar el comportamiento de la red de forma ágil y barata.

El objetivo principal de este trabajo será la implementación de una red de área local extensa mediante la utilización de SDN, que define un cambio de paradigma que permite reprogramar la red de forma centralizada y flexible.

Más específicamente, se pretende diseñar algoritmos para reducir el impacto de las tormentas de broadcast, programando varios filtros en redes SDN, permitiendo desplegar redes de área local más amplias, adecuadas para los requisitos de redes corporativas. Concretamente, se abordarán los procedimientos de autoconfiguración de IPv6 y los procedimientos de resolución de direcciones IP y físicas en IPv4 e IPv6. Para ello, se programará el controlador de la red SDN. El controlador de la red SDN es el "cerebro."encargado de comunicar a los conmutadores de la red qué deben hacer con cada flujo de paquetes nuevo.

Para la programación de la aplicación del controlador, se utilizará OpenDayLight, un controlador de redes SDN ampliamente extendido. El filtrado diseñado se programará en Java, lenguaje de programación orientado a objetos estudiado previamente en el Grado. Por otra parte, la herramienta básica para trabajar con SDN es el emulador Mininet, que permite crear redes virtuales junto con todos sus elementos en una sola máquina, facilitando la posterior interacción con dichas redes mediante líneas de comandos.

Como antecedentes a este proyecto, se encuentran estudios que presentan la limitación de las redes Ethernet en cuanto a escalabilidad debido a los protocolos de arranque. Especialmente, destacan los protocolos Address Resolution Protocol (ARP) e Internet Control Message Protocol version 6 (ICMPv6), ya que son los que mayor cantidad de tráfico generan.

En este trabajo se presenta el diseño de la solución que incluye las fases de detección e identificación de paquetes, filtrado y reenvío, para cada uno de estos protocolos pero capaz de funcionar simultáneamente para ambos. Este diseño se ha traducido en la implementación de un mecanismo para reducir el tráfico ARP y otro para el tráfico ICMPv6, que se ejecutan concurrentemente.

Por último se ha realizado una evaluación sobre una plataforma de SDN emulada que presenta los resultados obtenidos en diferentes experimentos, que demuestran cómo la utilización de un controlador con la solución implementada supone una importante disminución del tráfico de la red y plantea la posibilidad de extender el diseño a nuevos protocolos con la intención de optimizar el funcionamiento de estas redes.

Implementation of a Wide Local Area Network using Software Defined Networking

Bárbara Valera Muros

Keywords: Software Defined Networking (SDN), Mininet, OpenDayLight (ODL), Java, Controller, Filtering, Broadcast, Address Resolution Protocol (ARP), Internet Control Message Protocol version 6 (ICMPv6)

Abstract

The use of Ethernet as a network technology in corporative networks is justified by its low cost and ease regarding configuration and maintenance. However, this kind of networks that interconnects thousands of devices, is not scalable, especially due to the broadcast radiation. The broadcast radiation is produced when several devices send packets to the multicast address of the network. This process not only consumes the resources of the networks, but it also reduces the devices throughput.

With SDN, today's static network can evolve into an extensible service delivery platform capable of responding rapidly to changing business, end-user, and market needs. Software Defined Networking (SDN) represents a new paradigm where the switches performance can be reprogrammed in a centralized way. This implies a reduction of the costs and eases the reconfiguration of the network performance. Thus, the network will adapt itself to the environment and provide a Quality of Service (QoS) appropriate to the network's state.

The main purpose of this research will be the implementation of a wide Local Area Network using SDN, which implies a new paradigm able to reprogram networks in a flexible and centralized way.

Specifically, algorithms to reduce the impact of the broadcast storms will be designed by programming the filtering on SDN, which allows the implementation of wider local area networks, appropriate to meet the corporative networks requirements. In particular, procedures for dealing with auto configuration of IPv6 and IP and hardware address resolution on IPv4 and IPv6 will be addressed. Thus, the controller of the SDN will be programmed. It is is considered the "brain" of the network since it is responsible of managing the traffic routing on the network through the switches.

Regarding the controller's application programming, OpenDayLight will be used as it is a SDN controller widely used. The filtering designed will be programmed in Java, the object-oriented programming language studied in the Degree. The main tool used to work with SDN is the network emulator Mininet, that creates a realistic virtual network, running real kernel, switch and application code, on a single machine.

Previous research shows the limitation of the Ethernet-based networks, mainly regarding to the scalability due to the bootstrapping protocols. Especially, Address Resolution Protocol (ARP) and Internet Control Message Protocol version 6 (ICMPv6), since these two protocols are responsible of a high amount of generated traffic.

Bearing this in mind, the design of the solution is introduced divided on three stages of detection and identification of packets, filtering and redirection for each of these protocols, being able to perform simultaneously for both of them.

Generally speaking, since the controller is responsible of receiving and transmitting all the data traffic to the network, it will have a filter together with a table to store all the information available regarding the nodes belonging the network. Thus, the controller will be able to reply itself ARP and ICMPv6 requests, which makes unnecessary redirecting those requests to the rest of the network. This will make the data traffic of the network decrease, increasing at the same time the performance of the system.

Finally, this paper includes an evaluation stage presenting the results obtained in different experiments, which proves how the utilization of a controller with the solution implemented means a notable reduction of the data traffic in the network and proposes the possibility of extending the design to new protocols with the purpose of networks throughput optimization.

Yo, **Bárbara Valera Muros**, alumno de la titulación INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Bárbara Valera Muros

Granada a 1 de Julio de 2016.

D. **Juan José Ramos Muñoz**, Profesor del Área de Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Jorge Navarro Ortiz**, Profesor del Área de Comunicaciones del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Implementación de red de área local extensa mediante Software Defined Networking***, ha sido realizado bajo su supervisión por **Bárbara Valera Muros**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 1 de Julio de 2016.

Los directores:

Juan José Ramos Muñoz **Jorge Navarro Ortiz**

Agradecimientos

A mi familia, porque sin ellos no habría sido posible. A Rafa, por toda su ayuda, dentro y fuera de la Universidad. Y por supuesto a Juanjo y a Jorge, por su dedicación y apoyo. Gracias a todos.

"We don't make art, we solve problems."
- Eames

Índice general

Referencia de siglas y acrónimos

AD-SAL API-Driven Service Abstraction Layer

API Application Programming Interface

ARP Address Resolution Protocol

CAPEX CAPital EXpenditures

DHCP Dynamic Host Configuration Protocol

DHT Distributed Hash Tables

EoR Ethernet-over-Radio

EPC Evolved Packet Core

FSDM Floodless Service Discovery Model

IANA Internet Assigned Numbers Authority

ICMP Internet Control Message Protocol

IDE Integrated Development Environment

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IT Information Technology

JDK Java Development Kit

LAN Local Area Network

MAC Media Access Control

MD-SAL Model-Driven Service Abstraction Layer

MTC Machine Type Communication

ODL OpenDaylight

OPEX OPerating EXpense

OS Operating System

OSI Open System Interconnection

POM Project Object Model

QoS Quality of Service

RADVD Router Advertisement Daemon

REST Representational State Transfer

SAL Service Abstraction Layer

SDN Software Defined Networking

SFC Service Function Chaining

SLAAC StateLess Address AutoConfiguration

SSD Solid-State Drive

TCP Transmission Control Protocol

TTL Time To Live

UDP User Datagram Protocol

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo se realizará una contextualización del trabajo, así como la presentación de los conceptos clave del mismo. A lo largo de los diferentes apartados, se presentará el problema que se pretende resolver y una visión general de la solución alcanzada, facilitando así la comprensión del resto del trabajo, cuya estructura quedará planteada también en esta sección.

1.1. Contexto y motivación

Los servicios de datos móviles se han convertido poco a poco en imprescindibles para la mayoría de usuarios, con el consecuente éxito de los terminales inteligentes y la comunicación Machine Type Communication (MTC), o "comunicaciones máquina a máquina". Esto supone un incremento del tráfico en las redes inalámbricas, cuyo pronóstico [4] para el año 2020, mostrado en la figura 1.1, representa uno de los mayores retos a los que cualquier red de comunicación tendrá que enfrentarse en el futuro.

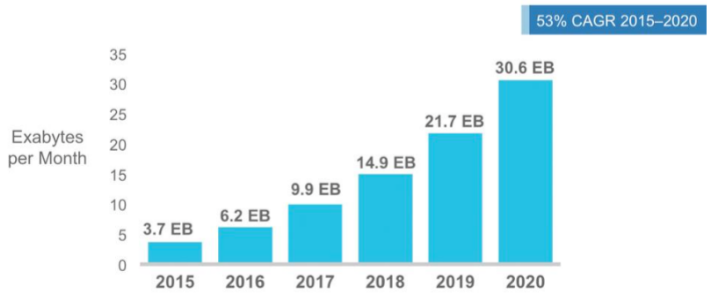


Figura 1.1: Cisco pronostica 30.6 Exabytes al mes de tráfico de datos móviles para 2020. Fuente: Cisco VNI Mobile, 2016.

A esto se le sumaría la reducción de la latencia y los costes asociados, de forma que se plantean nuevos diseños y arquitecturas de red con el fin de satisfacer las crecientes exigencias de los usuarios. El despliegue de red para las nuevas generaciones móviles requerirá un mejor servicio de portabilidad e interoperabilidad, lo que se traduce en una necesidad de adaptación de las redes a los nuevos dispositivos, suponiendo a su vez una mayor demanda de servicios. En la figura 1.2 se muestra dicho crecimiento.

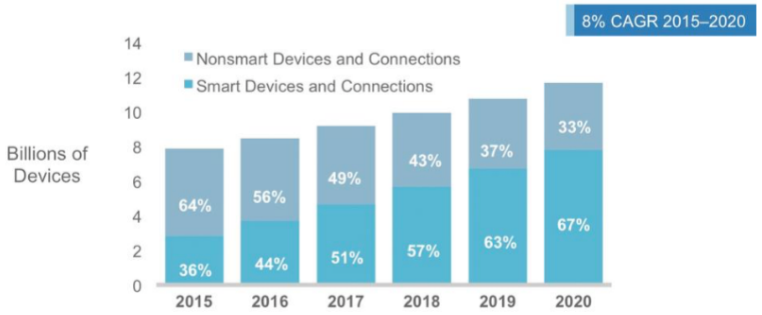


Figura 1.2: Crecimiento global de dispositivos y conexiones móviles inteligentes. Fuente: Cisco VNI Mobile, 2016.

Tradicionalmente, la mayoría de redes convencionales disponen de un diseño jerárquico, que tenía sentido cuando dominaba el paradigma Cliente-Servidor. Sin embargo, se trata de una arquitectura estática incapaz de satisfacer las necesidades de los entornos actuales. Algunas de las claves [5] para enfocar la búsqueda de un nuevo paradigma son:

- **Heterogeneidad en los patrones de tráfico:** Al contrario que como ocurría en los sistemas Cliente-Servidor, las aplicaciones modernas crean tráfico máquina a máquina mediante accesos a bases de datos y servidores, antes de devolver los datos obtenidos al usuario final. Además, los usuarios están cambiando los patrones de tráfico, al poder conectarse desde cualquier punto en cualquier momento en determinadas redes inalámbricas.

- **La influencia del mercado en las Information Technology (IT), en español tecnologías de la información:** Los usuarios han incrementado el número de dispositivos que utilizan para acceder a la red y los administradores de red tratan de acomodarse al uso de estos dispositivos a la vez que protegen la propiedad intelectual y corporativa de los usuarios y cumplen las normas establecidas.

- **El aumento de los servicios de cloud:** Debido principalmente a la gran acogida por parte de las empresas en el plano público y privado,

junto a la necesidad actual de aumentar la agilidad de acceso a aplicaciones, infraestructuras y otros recursos de telecomunicaciones y la complejidad de mejorar la seguridad de estos servicios, requiere una escalabilidad dinámica de la capacidad de cómputo, almacenamiento y recursos de red.

- **Big data y el aumento de ancho de banda requerido:** Lo que supone un procesamiento masivo por parte de miles de servidores, que deben estar directamente conectados entre sí, así como una demanda constante de ancho de banda a los proveedores de red de forma que se mantengan las conexiones permanentemente.

Software Defined Networking (SDN) surge como una de las posibilidades para suplir esa creciente demanda, considerándose una opción dinámica, gestionable, económica y adaptable. Además, reduce los costes asociados a las redes, conocimos como CAPital EXpenditures (CAPEX) y OPerating EXpense (OPEX), lo que se suma a las ventajas de utilizar esta arquitectura a la hora de renovar las redes de comunicación. En el capitulo 3.1 se realizará un estudio más detallado del estado del arte en el que se profundizará en esta arquitectura. Además, se redundará en los motivos por los que se está convirtiendo en una de las principales opciones para la modernización y desarrollo de futuras redes, siendo la apuesta actual de grandes compañías como Google, Orange o Verizon.

De este contexto se puede deducir la motivación para la realización del trabajo. El aprovechamiento de las redes es un sector en el que se ha invertido gran cantidad de recursos; pero los requerimientos de las telecomunicaciones son cada vez mayores, por lo que un salto de generación móvil implicaría un cambio completo de la red que suponga una solución definitiva y no temporal, como se ha hecho hasta ahora. Así, se encuentra la necesidad de una arquitectura de red que satisfaga las demandas actuales y futuras de proveedores, empresas y usuarios, planteándose SDN como una respuesta a esa necesidad. Y no sólo para redes móviles, sino también para redes corporativas, típicamente desarrolladas sobre el estándar Ethernet. [6]

1.2. Descripción del problema y visión general de la solución

A continuación, se presenta la problemática afrontada en este trabajo y una primera visión de lo que será la solución implementada posteriormente. Como ya se ha comentado, el aprovechamiento de las redes actuales es una solución temporal, por lo que la prioridad a la hora de modernizar las infraestructuras de telecomunicaciones debe ser diseñar una arquitectura que

se adapte a las futuras demandas a la vez que supere las limitaciones que encuentran las redes actuales.

En [1], se presenta una posible arquitectura de Ethernet sobre Radio, Ethernet-over-Radio (EoR), para la próxima generación móvil basada sobre Ethernet, optimizada para Internet Protocol version 6 (IPv6) y donde SDN es la clave para resolver los retos previstos. Con esta visión, esquematizada en 1.3, se pretende eliminar parte de la complejidad Evolved Packet Core (EPC), de forma que 5G sea más escalable y eficiente.

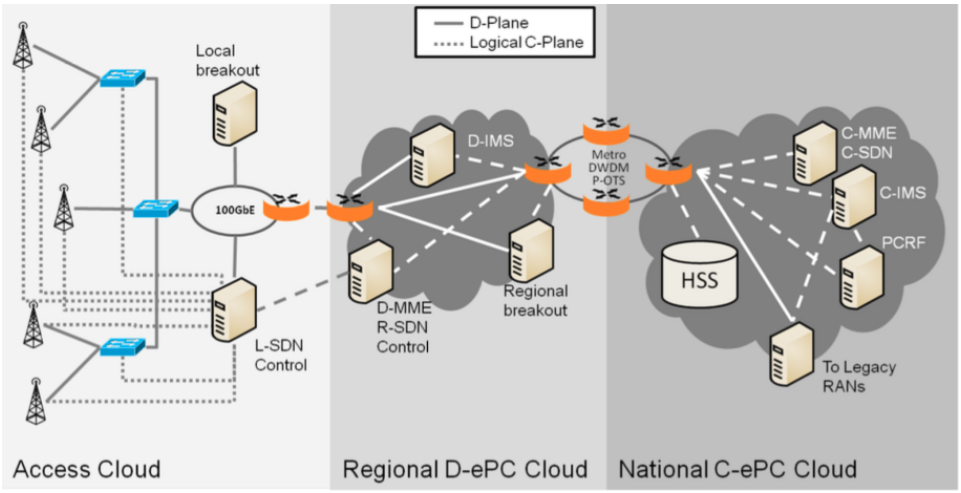


Figura 1.3: Arquitectura propuesta en [1] para la red 5G EPC

Sin embargo, la escalabilidad de las redes Ethernet está limitada [7] por las inundaciones de red (o tormentas de broadcast), causadas por los protocolos de arranque que utilizan los usuarios finales, como Address Resolution Protocol (ARP) [8] o Dynamic Host Configuration Protocol (DHCP) [9]. Para superar estas limitaciones, las nuevas arquitecturas basadas en Ethernet tratan de reducir las inundaciones de las tramas broadcast a la vez que ofrecen los servicios Ethernet esperados. Dichas tormentas son una forma de transmisión en la que un nodo emisor envía información a multitud de nodos receptores de manera simultánea, lo que implica la saturación de la red debido a la emisión constante de paquetes.

Por otra parte, StateLess Address AutoConfiguration (SLAAC) [10] permite la autoconfiguración de los hosts IPv6, tal y como se expone en la sección 3.3.2, lo que a su vez supone multitud de solicitudes multicast.

Así, el principal problema planteado es la disminución de la eficiencia de estos sistemas al producirse una inundación. A modo de solución, se presenta en este trabajo el filtrado de mensajes, de forma que el controlador disponga de unas tablas identificativas para cada nodo y sea el encargado de reenviar los mensajes dirigiéndolos a un destinatario limitado directamente,

de manera que se eviten las inundaciones de red siempre que sea posible.

Dicho filtrado se realizará inicialmente para el protocolo DHCP, de forma que sea escalable a otros protocolos para futuras implementaciones. Así, el controlador actuará como un switch inteligente capaz de aprender no solo las direcciones de los nodos implicados en el intercambio de mensajes, sino también su papel en dicho intercambio, identificando los diferentes agentes que intervienen para posteriormente dirigir los mensajes del protocolo en cuestión y evitar la sobrecarga de la red, mejorando en definitiva la eficiencia del sistema.

1.3. Metodología

En esta sección se expone la sistemática seguida en el desarrollo del trabajo. En primer lugar, se realizó un estudio bibliográfico de las propuestas para la red 5G, así como de SDN, ya que se presentaba como una de las apuestas de mayor relevancia.

Se realizó entonces un proceso de familiarización con las herramientas necesarias para trabajar con SDN. En concreto, se realizó un estudio del protocolo utilizado, OpenFlow; el controlador, OpenDaylight (ODL); y el emulador de redes, Mininet, profundizando con este último en la creación y análisis de diferentes topologías.

Una vez definidos los escenarios sobre los que aplicar la solución, se diseñó un algoritmo para la detección e identificación, filtrado y reenvío de paquetes en el controlador SDN. Posteriormente, se realizó la programación del código a partir de un switch inteligente. Para ello, se tuvieron en cuenta los mensajes del protocolo a filtrar, así como la consideración de IPv6 como la versión sobre la que realizar dicho filtrado. Esta parte compone el grueso del trabajo, ya que incluye el diseño de la solución y su posterior implementación.

Por último, se realizó una evaluación de la solución implementada teniendo en cuenta las mejoras en el rendimiento del sistema, así como las posibles vías de aplicación futuras.

1.4. Estructura de la memoria

Este apartado ilustra la organización de la memoria, de forma que se facilite la lectura y localización de las diferentes secciones que la componen.

- **Capítulo 1 - Introducción.** En este primer capítulo se contextualiza el trabajo, presentando los aspectos que han motivado la realización del mismo. Se expondrá también una breve descripción del problema abor-

dado, una visión general de la solución implementada, la metodología seguida y las principales fuentes consultadas, tratando de introducir al lector los conceptos básicos del trabajo realizado.

- **Capítulo 2 - Análisis de objetivos.** Este capítulo incluye los objetivos marcados para la realización del trabajo, así como los requisitos y aspectos formativos previos a dicha realización. Por último, se realizará un estudio del alcance del proyecto plasmando las aportaciones y logros alcanzados.

- **Capítulo 3 - Estado del arte.** Se realiza un estudio del estado del arte, mencionado brevemente en el capítulo introductorio. Se presentan por tanto en este capítulo los conceptos teóricos a seguir en el desarrollo práctico del trabajo, profundizando en SDN, IPv6 y las herramientas utilizadas.

- **Capítulo 4 - Recursos y planificación.** En este capítulo se detalla la planificación temporal del desarrollo del trabajo, describiendo las fases en las que se divide su realización. Además, se analizan los recursos necesarios, concluyendo con una estimación del coste del proyecto.

- **Capítulo 5 - Resolución del trabajo.** Este capítulo se centra en el desarrollo del trabajo en sí, desde los métodos inicialmente propuestos hasta la implementación final, pasando por las diferentes fases de diseño de la solución, en las que se detallan los métodos que componen el programa implementado.

- **Capítulo 6 - Evaluación de la solución.** Este capítulo incluye todo el proceso de evaluación de los resultados obtenidos, planteando la fase de pruebas a la que se sometió la solución desarrollada y los resultados obtenidos en dicha fase.

- **Capítulo 7 - Conclusiones y proyección de futuro.** Por último, se exponen las conclusiones alcanzadas tras la finalización del trabajo, en referencia tanto a los resultados obtenidos como a las herramientas utilizadas. Así mismo, se incluye una proyección futura de las posibles vías de investigación en las que aplicar el trabajo y una valoración personal para finalizar.

- **Apéndices.**

  - **Instalación y puesta en marcha de las herramientas.** Se detalla la configuración del entorno de desarrollo, incluyendo las guías para la instalación de IPv6 y ODL.
  - **Problemas frecuentes.** Se profundiza en los problemas afrontados, principalmente para la configuración inicial y respecto a

las diferentes versiones del controlador existentes, así como la compatibilidad entre los diferentes sistemas operativos.

- **Bibliografía.** Se incluyen las fuentes consultadas, tanto para la realización práctica del trabajo como para la redacción de la memoria.

1.5. Bibliografía relevante

A continuación, se presentan las principales fuentes consultadas durante la realización de este trabajo, destacando las referencias de mayor importancia en cada uno de los campos abordados.

- **Arquitectura propuesta.**

  Principalmente artículos del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

  [7] Ameigeiras, P.; Ramos-Muñoz, J. J.; Schumacher, L.; Prados-Garzon, J.; Navarro-Ortiz, J. and Lopez-Soler, J. M. "Link-level access cloud architecture design based on SDN for 5G networks", IEEE Network, 2015

  [1] Cattoni, A. F.; Mogensen, P. E.; Vesterinen, S.; Laitila, M.; Schumacher, L.; Ameigeiras, P. and Ramos-Muñoz, J. J. "Ethernet-based mobility architecture for 5G Cloud Networking (CloudNet)", 2014 IEEE 3rd International Conference on, 2014

- **SDN y ODL.**

  Definiciones y motivación de SDN y manual de desarrollador para OpenDaylight.

  [5] Open Networking Foundation. "Software-Defined Networking: The New Norm for Network", ONF White Paper, 2012

  [11] Linux Foundation. "OpenDaylight Developer Guide", 2015

- **IPv6 y Mininet.**

  Especificación del protocolo y página oficial de la herramienta.

  [12] Deering, S. and Hinden, R."RFC 2460 Internet Protocol, Version 6 (IPv6) Specification", 1998

  [13] Lantz, B.; Handigol, N.; Heller, B. and Jeyakumar, V. "Introduction to Mininet", 2015

Capítulo 2

Análisis de objetivos y alcance del proyecto

En este capítulo se abordan las cuestiones previas al diseño y la implementación del trabajo. A continuación, se presentan los diferentes objetivos planteados al inicio del proyecto, seguidos por los requisitos y aspectos de formación previos necesarios para su consecución, así como un análisis del alcance final del trabajo realizado.

2.1. Objetivos

El uso de Ethernet como tecnología de red para redes empresariales se justifica por su bajo coste y facilidad de configuración y mantenimiento. Sin embargo, este tipo de redes no son muy escalables, debido en parte a las tormentas de broadcasts, que afectan al rendimiento de los dispositivos. El objetivo principal de este trabajo será la implementación de una red de área local extensa mediante la utilización de SDN, que define un cambio de paradigma que permite reprogramar la red de forma centralizada y flexible.

Se pretende por tanto desarrollar procedimientos SDN que permitan reducir el problema de los broadcasts para mejorar la escalabilidad de estas redes. Concretamente, los objetivos de este trabajo son:

1. Revisión bibliográfica de SDN y escalabilidad de redes.

2. Instalación y configuración el emulador de redes SDN Mininet.

3. Puesta en marcha del emulador Mininet y del controlador OpenDayLight.

4. Diseño y evaluación de servicios de filtrado de mensajes de broadcast sobre SDN.

Se podría redefinir por tanto el objetivo principal del trabajo como la implementación de una red con proyección de futuro, teniendo en cuenta las limitaciones de las redes y protocolos actuales y considerando no sólo los requisitos para su renovación sino también las tecnologías emergentes de las que disponemos para dicho avance.

2.2. Requisitos

Esta sección describe los requisitos básicos para el correcto desarrollo del trabajo, así como los conocimientos previos necesarios para su realización.

Principalmente, se requieren nociones de fundamentos de redes, ya que serán la base del trabajo. Además, es necesario manejar conceptos de escalabilidad y filtrado, de forma que se implementen de forma óptima en el algoritmo diseñado, así como de protocolos, especialmente IPv6. Será también indispensable la programación orientada a objetos, en concreto con Java, que es el lenguaje elegido para el diseño del controlador. Adicionalmente, será necesario un estudio en profundidad de las redes SDN, así como de los diferentes controladores disponibles, con intención de escoger el que mejor se adecue a los propósitos de este trabajo.

Por otra parte, respecto a las necesidades de Hardware, bastará con un ordenador para la realización práctica. El Software se compone de Java Development Kit (JDK) y Mininet, herramientas básicas para trabajar con SDN. Dado que el Operating System (OS), en español Sistema Operativo, que mejor se adapta al emulador Mininet es Ubuntu, se realizará una instalación de la última versión disponible en una máquina virtual, de forma que las conexiones externas del ordenador utilizado no interfieran en el análisis del tráfico de la red a implementar. Mininet permite crear fácilmente redes virtuales, y los escenarios a tener en cuenta se pueden programar con Python, por lo que en caso de realizar pruebas sobre topologías específicas será necesario conocer este lenguaje. Inicialmente, con una topología básica será suficiente, por lo que bastará con familiarizarse con la herramienta y las posibilidades ofrecidas por defecto. El controlador ODL también requiere instalación y configuración. En el capítulo 3.2 se proporciona más información acerca del emulador y la compatibilidad entre sistemas. Dicha compatibilidad, junto con la actualización continua de la versión del controlador utilizada, supuso una serie de problemas, detallados en el apéndice B.

2.3. Alcance del proyecto

Por último, se describe el alcance de este trabajo. En primer lugar, se trata de un estudio sobre las redes SDN y la posibilidad de integrarlas como arquitectura sobre la que desarrollar futuras redes de telecomunicaciones. Mediante el estudio del estado de arte, se profundiza en la situación actual de SDN, así como en las herramientas de las que se dispone para su manejo. Se emulará una red SDN y se evaluará el rendimiento de la misma, tomando como referencia las limitaciones de las redes actuales.

Así, se diseñará un algoritmo capaz de integrar los requerimientos de la red de forma óptima, que se aplicará sobre el controlador ODL una vez implementado. Para ello, será necesario estudiar la programación para redes SDN, así como el modo de evaluar el algoritmo desarrollado.

El desarrollo de esta implementación se aplicará en el controlador, que se encargará de filtrar los paquetes y redirigirlos a los diferentes agentes implicados en la comunicación, con el fin de optimizar el rendimiento del sistema.

Generalmente, debido al protocolo en cuestión utilizado, se generarán ciertos mensajes que se distribuyen mediante inundaciones al resto de la red al completo. Se considera por tanto que el mayor reto afrontado en la implementación de la red será mantener un cierto rendimiento a pesar de dichos mensajes, por lo que el filtrado de los mismos será una de las bases de la implementación.

Además, ya que se plantea una posible arquitectura para generaciones móviles futuras, se ha tenido en cuenta que la versión del Internet Protocol (IP) utilizada sea la 6. IPv6 será la versión encargada de sustituir a Internet Protocol version 4 (IPv4), actualmente implementada en la mayoría de dispositivos que acceden a Internet, ya que IPv4 limita el número de direcciones de red admisibles, restringiendo por tanto el crecimiento de Internet y su propio uso.

Así, se publicó a principios de 2010 que las direcciones IP sin asignar restantes no superaban el 10 % [14]. Posteriormente, a principios de 2011, la Internet Assigned Numbers Authority (IANA), en español Agencia Internacional de Asignación de Números de Internet, confirmó que el último bloque de direcciones disponibles había sido entregado a la organización encargada de asignar direcciones IP en Asia. En el capítulo 3.3.2 se profundiza en este protocolo, así como en las motivaciones para su implementación, no sólo debida a la limitación de dispositivos impuesta por IPv4. Por esto, otro de los objetivos del trabajo será adaptar a IPv6 la implementación, de forma que tenga aplicación real en un futuro.

Respecto al entorno en el que se realizará el trabajo, SDN se considera

una de las apuestas más relevantes dentro de las tecnologías emergentes para el desarrollo de las telecomunicaciones, por lo que otro de los objetivos planteados es el análisis de dicha arquitectura y la familiarización con las herramientas destinadas al manejo de la misma. Se realizará por tanto una implementación sobre controlador real, así como emulaciones con Mininet, de forma que la solución se ejecute en un entorno de hasta 20 ordenadores emulados.

Capítulo 3

Estado del arte

En este capítulo se realiza un análisis exhaustivo de las tecnologías implicadas en el proyecto. Concretamente, se profundiza en SDN, base en este trabajo, así como en el protocolo Openflow y el controlador ODL. Se realizará un estudio también de las herramientas utilizadas y de los protocolos sobre los que se realiza la implementación de la solución desarrollada, justificando cada uno de ellos y realizando una comparativa con el resto de opciones descartadas para dicho desarrollo.

3.1. Software Defined Networking

En primer lugar, la motivación de utilizar SDN en lugar de otra tecnología reside en que las arquitecturas de red de las que disponemos actualmente no fueron diseñadas para cumplir con las necesidades actuales de los usuarios y empresas, y los diseñadores de red encuentran ciertas limitaciones [15] a destacar en estas redes:

- **Complejidad.** Con el fin de acomodar las redes a las necesidades de los usuarios, se definen multitud de protocolos en aislamiento, de manera que realicen funciones individuales y enfocados a problemas específicos, por lo que no se benefician de la abstracción en actividades conjuntas.
- **Políticas incoherentes.** Para implementar una política que abarque a la red completamente, los administradores de red deben configurar miles de mecanismos y dispositivos. La complejidad de las redes dificulta la aplicación de políticas de acceso, seguridad y calidad de servicio, o Quality of Service (QoS), lo que supone, entre otras consecuencias, el incumplimiento de regulaciones y la vulnerabilidad ante brechas en la seguridad.

- **No escalable.** A la vez que las demandas de datos aumentan rápidamente, la red debe crecer de la misma forma. Sin embargo, la red se vuelve más compleja con la suma de cientos de miles de dispositivos de red que deben ser configurados y gestionados.
- **Dependencia del vendedor.** Las nuevas capacidades y servicios perseguidos por proveedores y empresas en respuesta a las necesidades dinámicas de los negocios y la demanda de los clientes se ven frenadas por los ciclos de producción de los equipamientos por parte de los vendedores, que pueden abarcar hasta más de tres años. La ausencia de un estándar y las interfaces libres limitan la capacidad de los operadores de adaptar la red a su entorno particular. Esta incompatibilidad entre requerimientos y capacidad de red deriva en un punto de inflexión, en el que se crea la arquitectura SDN y se desarrollan estándares asociados.

Como respuesta a estas necesidades, surgen las redes SDN, que permiten atender las necesidades de los usuarios de forma dinámica y escalable, mediante una programación de redes centralizada que evita al administrador gestionar los servicios requeridos a bajo nivel, de forma que la red se adapta al entorno, ofreciendo una calidad de servicio acorde al estado del mismo.

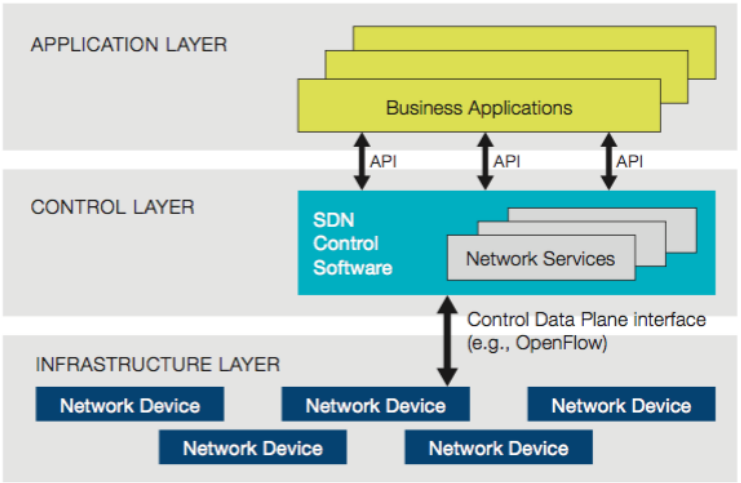


Figura 3.1: Arquitectura simplificada de SDN

Así, estas redes ofrecen beneficios como la reducción del CAPEX, gracias a la reutilización de hardware existente, que reduce la necesidad de invertir en nuevo; y del OPEX, mediante un control algorítmico de los elementos de la red, que al ser más programables agilizan su configuración y gestión, de forma que no sólo disminuye el tiempo invertido en gestión por los administradores, sino también la probabilidad de error humano.

Se proporciona también mayor agilidad y flexibilidad, permitiendo así un despliegue de aplicaciones, servicios e infraestructuras para alcanzar satisfactoriamente los objetivos en el menor tiempo posible. Además, la creación de nuevas aplicaciones y modelos de negocio permite una innovación que aumenta el valor de las redes en sí.

Como contextualización [16], el despliegue de Internet dio lugar a la necesidad de la estandarización de protocolos que soportaran las aplicaciones en desarrollo, lo que provocó que se plantearan alternativas para el control de redes, entre las que se encuentra la reprogramación de las mismas. Surgen entonces las redes activas, dirigidas al control de red a partir de una interfaz programable con capacidad para mostrar los recursos de cada nodo individualmente y aplicar acciones para un grupo de paquetes definido, que recibe el nombre de flujo.

A su vez, el aumento del tráfico en Internet, junto con la necesidad de redes manejables, predecibles y confiables, derivó en el análisis de la ingeniería de tráfico, hasta entonces limitada por los protocolos de enrutamiento convencionales. Normalmente, dicho enrutamiento se realizaba mediante routers y switches, con unos recursos limitados, lo que hacía al sistema perder capacidad. Como solución, se plantea el concepto de separación del plano de control de red (software) y del plano de datos (hardware que conmuta los paquetes en la red), base de las redes SDN, tal y como muestra la figura 3.1. [17]

Como muestra la figura 3.2, las aplicaciones usan la interfaz de programación, o Application Programming Interface (API) NorthBound, sobre el plano de control para reforzar sus principios en el plano de datos sin interactuar con el mismo directamente. La interfaz entre el plano de control y el de datos se apoya en SouthBound APIs, donde el controlador SDN usa dichas APIs para comunicarse con los equipos de la red en el plano de datos. [18] Dichos equipos deberán por tanto soportar las APIs estandarizadas en este nivel.

SDN posibilita la administración de la red al completo a través de un sistema inteligente que permite la asignación de recursos según la demanda, redes virtualizadas y servicios cloud seguros. Por tanto, la red estática evoluciona en una plataforma de servicio independiente capaz de responder rápidamente a las necesidades de mercado de los usuarios finales, lo que simplifica en gran medida el diseño y las operaciones de red. Consecuentemente, los dispositivos no tienen por qué entender y procesar por sí mismos males de estándares y protocolos, sino simplemente aceptar instrucciones de los controladores SDN.

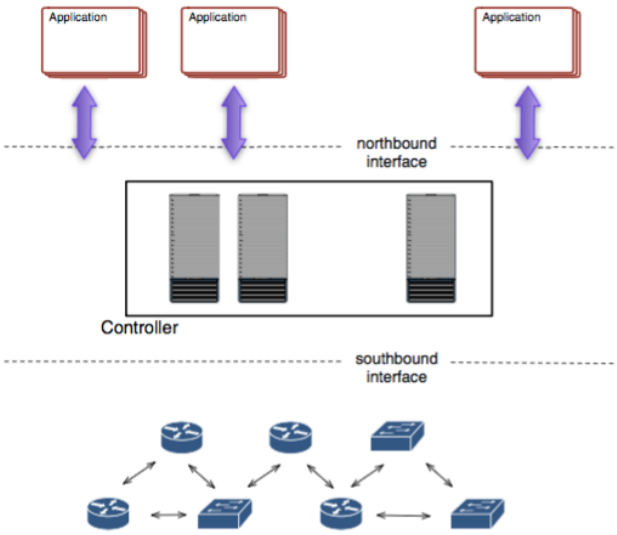


Figura 3.2: Estructura de SDN con las APIs

3.1.1. Ventajas de las redes SDN

A modo de definición, SDN es una arquitectura emergente caracterizada por ser dinámica, manejable, rentable y adaptable, lo que la hace ideal para los requerimientos de gran ancho de banda de las aplicaciones actuales. Además, se destaca que estas redes son:

- **Directamente programables.** El control de red es directamente programable debido al desacoplo con el resto de funciones, gracias a la separación del plano de control y de datos.
- **Ágiles.** Abstraer el control del resto, de nuevo debido a la separación de los planos, permite a los administradores ajustar dinámicamente el tráfico de la red mediante flujos para adaptarse a las necesidades del sistema, a menudo cambiantes.
- **Gestionadas centralmente.** La inteligencia de la red se centra en los controladores que mantienen una visión global de la red, interactuando con el resto de elementos, lo que para las aplicaciones se muestra como un único switch lógico.
- **Configuradas mediante la programación automatizada.** Permite a los administradores de la red configurar, administrar, asegurar y optimizar los recursos de red rápidamente mediante programas dinámicos y automatizados, que pueden escribir los propios administradores, ya que no dependen del software propietario.

- **Redes basadas en estándares abiertos.** Lo que permite la simplificación del diseño y las operaciones de red, ya que las instrucciones vienen dadas por los controladores y no dependen de proveedores específicos, protocolos propietarios o múltiples dispositivos.

Todas estas características, así como los beneficios teóricos mencionados previamente, justifican el uso de SDN, que gracias a las NorthBound APIs permite el desarrollo de nuevas aplicaciones sin necesidad de modificar la estructura de la arquitectura, reduciendo además el coste inicial de instalación y equipamiento gracias a las SouthBound (comunican los elementos a más bajo nivel con el controlador). Cabe añadir en esta justificación el apoyo por parte de la industria del sector, en especial de grandes compañías como Google [19], Verizon [20] y Orange [21], que además de respaldar esta arquitectura han conseguido reducir la complejidad de sus redes, destacándose como casos de éxito.

3.1.2. Protocolo OpenFlow

OpenFlow es un protocolo de comunicaciones diseñado para dirigir el manejo y enrutamiento del tráfico en una red conmutada. Es una de las alternativas de las que dispone SDN para gestionar la red, y dado que se trata de un estándar abierto, se ha convertido en el modelo estándar de implementación de SDN. Se ha traducido así como la primera interfaz de comunicaciones definida entre las capas de control y de transporte en esta arquitectura. [22]

OpenFlow puede considerarse además un compromiso pragmático que permite a los investigadores realizar experimentos en redes con routers y switches heterogéneas de forma uniforme, sin la necesidad de que los proveedores especifiquen el funcionamiento interno de sus productos o que los investigadores desarrollen software de control específico para dichos productos. [23]

Como alternativas a OpenFlow, se podría destacar el estándar desarrollado por Cisco OpFlex [24], que permite la aplicación de políticas de tráfico a través de switches y routers físicos y virtuales en sistemas con elementos de red de diferentes proveedores. Este estándar descentraliza la red, de forma que se reduzca su dependencia con el controlador, ya que es considerado un cuello de botella en las redes programables. De esta forma, se definirían una serie de políticas en el controlador, que mediante OpFlex se comunicarían a los elementos inteligentes de la red, capaces de interpretar dichas políticas y adaptarlas a la situación. Sin embargo, dicha interpretación requiere de un agente embebido en los elementos que forman parte de la red.

Como ventaja sobre OpenFlow, por tanto, se presenta la independencia

del controlador en la virtualización y programación de la red, mientras que la necesidad de añadir un agente embebido en cada agente de la red se considera la principal desventaja en cuanto a los costes del sistema. Por otra parte, en relación a la industria del sector, OpenFlow está más respaldado ya que OpFlex ha sido desarrollado por Cisco, aunque ambos son abiertos.

El diseño del protocolo se apoya sobre tres bases: los switches con soporte para OpenFlow, las tablas de flujos instaladas en dichos switches para la gestión del tráfico y el controlador encargado de comunicar a los switches la información necesaria para administrar el tráfico de la red. En la figura 3.3 se muestra dicho proceso de comunicación, siendo el controlador el encargado de añadir y eliminar flujos y el switch el de encaminar los paquetes.

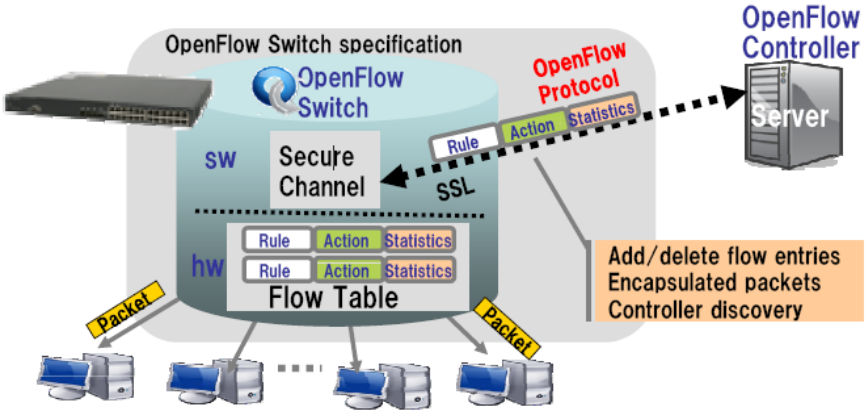


Figura 3.3: Esquema de comunicación en el protocolo OpenFlow. Extraída de [25]

Así, aunque el switch OpenFlow es responsable del reenvío de paquetes, las decisiones de enrutamiento serán tomadas por el controlador, comunicándose ambos a través del protocolo OpenFlow, que define los mensajes que hacen referencia a los paquetes enviados, recibidos, la identificación de estados y la modificación de las tablas de flujos para el encaminamiento, de forma que cuando el switch recibe un paquete para el que no tiene entradas en la tabla de flujo, se lo reenvía al controlador, que será el encargado de decidir si el paquete es descartado o si se agregará una entrada en las tablas para que el switch pueda gestionarlo por sí mismo en el futuro, en caso de volver a recibir un paquete similar.

Una vez el switch conoce la dirección del controlador, se inicia una conexión TCP estándar con el mismo. Tal y como muestra la figura 3.4, los switches OpenFlow constan de tablas de flujo, en las que se almacena la información asociada a cada paquete, y de un canal OpenFlow seguro que conecta el switch con el controlador mediante este protocolo. Básicamente, en el momento que un paquete llega al switch, éste podrá enviar el paquete

en cuestión a un puerto establecido para ese flujo, eliminar el flujo o encapsularlo y enviarlo al controlador por el canal seguro que se muestra en la imagen.

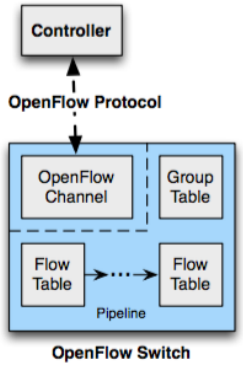


Figura 3.4: Estructura básica de un Switch OpenFlow

Se describen a continuación los principales componentes del protocolo: los puertos y las tablas de flujo.

Puertos OpenFlow

Son las interfaces de red definidas entre OpenFlow y el resto de la red, de forma que los switches del sistema dispongan de una conexión lógica, encontrándose así todos conectados para el envío de paquetes por estos puertos. Cabe diferenciar los puertos ingress, por los que se reciben los paquetes, y los puertos output, por los que se realiza el envío. Además, se pueden diferenciar tres tipos de puertos:

- **Físicos.** Corresponden a una interfaz hardware del switch y vienen definidos por el mismo.
- **Lógicos.** Son una abstracción de alto nivel, por lo que no tienen que corresponder necesariamente con una interfaz física del switch. Deben ser capaces de realizar encapsulamiento de paquetes y mapeo de varios puertos físicos. Su implementación debe ser transparente al protocolo OpenFlow, de forma que estos puertos sean capaces de comunicarse y realizar el procesado de paquetes como si se tratara de puertos físicos, con la diferencia de que se añade un campo llamado Tunnel-ID para la identificación del puerto.
- **Reservados.** Son puertos reservados a acciones de envío específicas:
  - ALL. A todos los puertos por los que el switch puede reenviar paquetes.

- CONTROLLER. Al controlador.
- TABLE. Al comienzo del pipeline de OpenFlow.
- IN_PORT. Al puerto por el que se ha recibido el paquete.
- ANY. A cualquier puerto no especificado.
- LOCAL. A la red local del switch.
- NORMAL. Al pipeline que no es OpenFlow.
- FLOOD. A todos los puertos excepto al de entrada.

Tablas OpenFlow

Las tablas de flujos OpenFlow están instaladas en cada uno de los switches e indicarán a cada dispositivo qué hacer con el tráfico, tal y como se expone a continuación.

Proceso de Pipeline. El proceso de pipeline en OpenFlow cuenta con una o más tablas de flujo para cada switch, y cada tabla contendrá a su vez múltiples entradas de flujos, definiéndose así la interacción de los paquetes con las propias tablas. Cada switch debe tener por tanto al menos una tabla de flujo, de forma que si éste fuera el caso, el proceso pipeline sería simple. En caso de que disponga de varias tablas, quedarán numeradas secuencialmente, empezando por 0. De esta forma, se comprobarán primero los flujos de la tabla 0 al iniciar el proceso pipeline, que seguiría el esquema siguiente, representado gráficamente en 3.5. Existe también una última tabla, *missing*, encargada de decidir qué hacer con los paquetes para los que no se encuentra coincidencia.

En primer lugar, se intenta hacer matching del paquete con cada flujo en la primera tabla, de forma que todos los campos del paquete coincidan con los valores marcados como match en ese flujo. Hacer matching implica consultar cada una de las tablas y en función de los resultados obtenidos aplicar las instrucciones definidas. Así, si en una tabla no hay ninguna entrada de flujo que coincida, se pasará a la siguiente tabla. En caso de que sí exista una coincidencia, el proceso se detiene en esa tabla y se realizan las acciones definidas en la entrada coincidente. Si la acción de la entrada es Goto-Table, el paquete será procesado por otra tabla, que deberá ser de orden superior a aquella con el flujo coincidente para evitar bucles.

En caso de que no haya matching en ninguna tabla, el paquete pasará a la tabla missing, que decidirá si debe inundar la red con el paquete, mandarlo al controlador o comenzar de nuevo con un matching más flexible, ignorando algunos campos. En caso de no existir la tabla missing, los paquetes se descartan automáticamente, mientras que si tienen un Time To

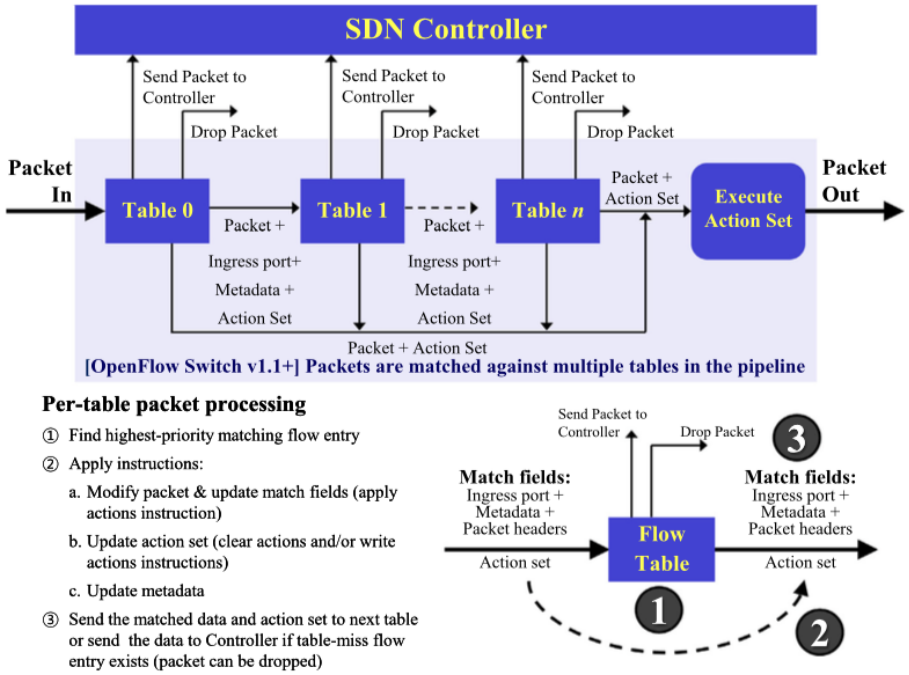


Figura 3.5: Flujo de paquetes en el proceso de Pipeline

Live (TTL) inválido son enviados al controlador directamente. Todo este proceso se muestra en la imagen 3.6.

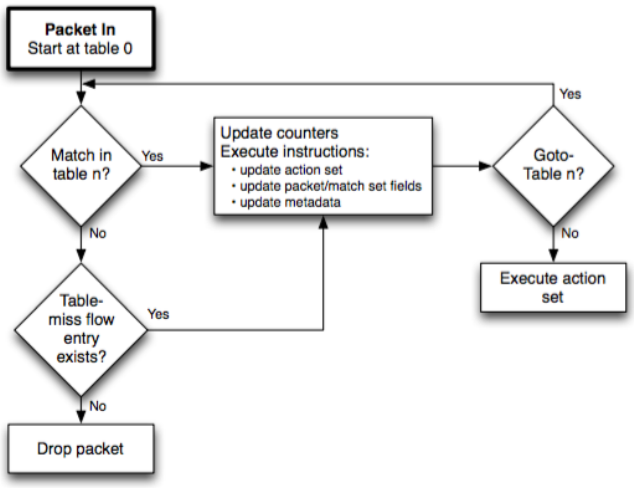


Figura 3.6: Diagrama de flujo detallando los flujos de los paquetes en un switch OpenFlow

Entradas de flujo. Las entradas de flujo especifican las acciones que se aplican a los paquetes con los que coinciden en el proceso de pipeline. Tal y como muestra la tabla 3.1.2, están compuestas por diferentes campos, a saber:

Campos Match	Prioridad	Contadores	Instrucciones	Timeouts	Cookie

Tabla 3.1: Componentes de una entrada de flujo.

- Campos Match. Para comparar el paquete y la entrada de flujo, contiene el puerto de entrada y las cabeceras del paquete. En la tabla 3.2 se muestran los posibles campos.

- Prioridad. Para comprobar la procedencia del matching. En caso de que sea 0 y los campos match estén vacíos, se tratará de la tabla missing.

- Contadores. Para actualizar los paquetes, teniendo en cuenta el número de entradas de la tabla, la duración de cada entrada o los paquetes transmitidos o recibidos por cada puerto.

- Instrucciones. Para indicar las acciones que realizará el switch una vez se realice matching del paquete con esa entrada de flujo.

- Timeouts. Para definir la validez de la entrada de flujo, pueden ser idle, si define un periodo de validez específico que se reiniciará cada vez que un paquete coincida con ese flujo, o hard, si ese período establecido es fijo. En ambos casos, una vez expire ese tiempo de validez, el flujo será eliminado de la tabla.

- Cookie. Para filtrar las estadísticas de flujos y su modificación y eliminación.

Campo	Descripción
OXM_OF_IN_PORT	Puerto de entrada.
OXM_OF_ETH_SRC	Dirección Ethernet de origen.
OXM_OF_ETH_DST	Dirección Ethernet de destino.
OXM_OF_ETH_TYPE	Tipo de paquete Ethernet
OXM_OF_IP_PROTO	Protocolo IPv4 o IPv6
OXM_OF_IPV4_SRC	Dirección origen IPv4.
OXM_OF_IPV4_DST	Dirección destino IPv4.
OXM_OF_IPV6_SRC	Dirección origen IPv6.
OXM_OF_IPV6_DST	Dirección destino IPv6.
OXM_OF_TCP_SRC	Puerto origen Transmission Control Protocol (TCP)
OXM_OF_TCP_DST	Puerto destino TCP
OXM_OF_UDP_SRC	Puerto origen User Datagram Protocol (UDP)
OXM_OF_UDP_DST	Puerto destino UDP

Tabla 3.2: Campos Match. Extraída de [26]

Por último, comentar que este protocolo está en continuo desarrollo, al igual que toda la tecnología relacionada con SDN, por lo que se encuentran diferentes versiones del protocolo, siendo la más reciente OpenFlow 1.5. Sin embargo, no todas las versiones están respaldadas por la industria y difieren en compatibilidad y estabilidad, factor a tener en cuenta a la hora de realizar el proyecto. En este trabajo, se utiliza la versión 1.3, que es la que mejores prestaciones proporciona y la última con soporte comercial.

3.1.3. Controladores SDN

El controlador es uno de los elementos principales en las redes SDN, considerándose el "cerebro" de la red, ya que se encarga del plano de control de la arquitectura. Al igual que con el protocolo, se pueden encontrar controladores de código abierto y controladores comerciales. Dado que se pretende minimizar el coste económico de este proyecto, se descarta el uso de un controlador comercial y se procede al estudio de las alternativas de código abierto. [27]

El primer controlador con soporte para OpenFlow fue NOX. Escrito en C++, se desarrolló para distribuciones de Linux. Posteriormente, surge POX a modo de relevo, desarrollado en Python con la intención de agilizar el desarrollo de software para el controlador. Aprender a desarrollar software en POX es rápido, razón principal para ser utilizado en investigación, demostraciones y experimentación; sin embargo la ejecución de la red es más lenta que con otros lenguajes de programación. Además, POX soporta únicamente la versión 1.0 de OpenFlow.

Otro controlador escrito en Python es Ryu, que soporta hasta la versión

1.4 del protocolo OpenFlow. A diferencia de POX, permite trabajar de forma distribuida, aunque de nuevo se presenta el problema del lenguaje de programación, que comparado con la ejecución de otros controladores puede considerarse su mayor desventaja.

ONOS, por otra parte, ofrece un sistema operativo de red distribuida, escrito en Java, lo que requiere mayor tiempo de aprendizaje a la hora de desarrollar aplicaciones. Como interfaz SouthBound, soporta las versiones 1.0 y 1.3 de OpenFlow, y ofrece buenos resultados de ejecución, además de ser distribuido.

Otro controlador es OpenDayLight, a partir de ahora ODL, que también es de código abierto, escrito en Java, robusto y proporciona un buen rendimiento de ejecución y soporte a nivel de producción. Como principal desventaja, se presenta la complejidad y el tiempo empleado en aprender a desarrollar aplicaciones. ODL soporta las versiones 1.0 y 1.3 de OpenFlow. Destaca, sin embargo, el gran apoyo que recibe por parte de la industria, tal y como muestra la figura 3.7, lo que ha resultado uno de los principales motivos por los que se eligió como controlador para el sistema desarrollado en este proyecto.

Controlador OpenDayLight

Debido al continuo desarrollo en el que se encuentra esta tecnología, se dispone de múltiples distribuciones de ODL. La primera fue Hydrogen, disponible desde Febrero de 2014, seguida por Helium, en Septiembre de ese mismo año. En Junio de 2015 se lanzó la primera versión de Lithium, actualizada por última vez en Marzo de 2016. En Febrero de 2016 salió la última distribución disponible del controlador, Beryllium, pero debido a que el proyecto se encontraba ya en fase de diseño, previa familiarización con las herramientas, y en concreto con la última versión de Lithium, se descartó la posibilidad de actualizarlo y se continuó trabajando con Lithium.

Con esta distribución se pueden componer arquitecturas propias de servicio o adaptar los servicios de red dinámicos en un entorno de nube, aplicar políticas de calidad de forma dinámica y aplicar virtualización con la tecnología Service Function Chaining (SFC), que permite utilizar de forma flexible y rápida varias funciones de red. [11]

Tal y como muestra la figura 3.8, la estructura del controlador muestra tres capas bien diferenciadas:

- **Aplicaciones de red, instrumentación y servicios.** Es la capa de más alto nivel, en la que se encuentran las aplicaciones de control y monitorización de la red.
- **Controlador.** Capa central, cuenta con una serie de módulos que

Figura 3.7: Miembros del proyecto OpenDayLight.

permite a la capa superior acceder a la información sobre el estado de la red.

- **Elementos virtuales y físicos de la red.** Capa inferior compuesta de los elementos programables de la red, que se comunicarán con la capa central mediante la interfaz SouthBound, encargada de que cualquier elemento sea compatible con ODL, sin importar el fabricante.

Respecto a su implementación, ODL está implementado sobre Java, por lo que su ejecución es independiente del OS siempre y cuando éste soporte Java, lo que hace este controlador compatible con la mayoría de OS disponibles. Además, la capa Service Abstraction Layer (SAL) separa el controlador de los elementos y protocolos de red, encargándose de traducir las peticiones de las capas superiores a los protocolos de estos dispositivos soportados por el controlador. Esta capa presenta dos arquitecturas, API-Driven Service Abstraction Layer (AD-SAL) y Model-Driven Service Abstraction

Layer (MD-SAL), según el tipo de abstracción que presente.

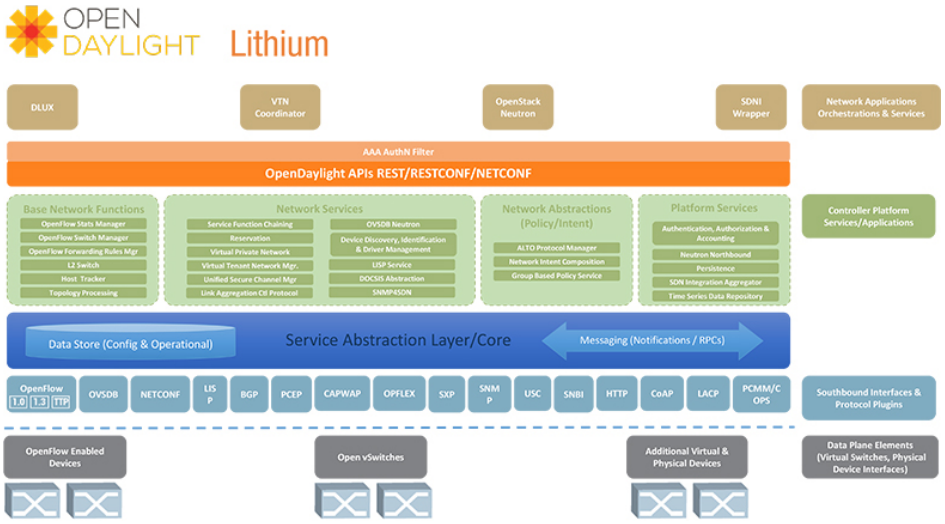


Figura 3.8: Estructura de la versión Lithium de OpenDayLight

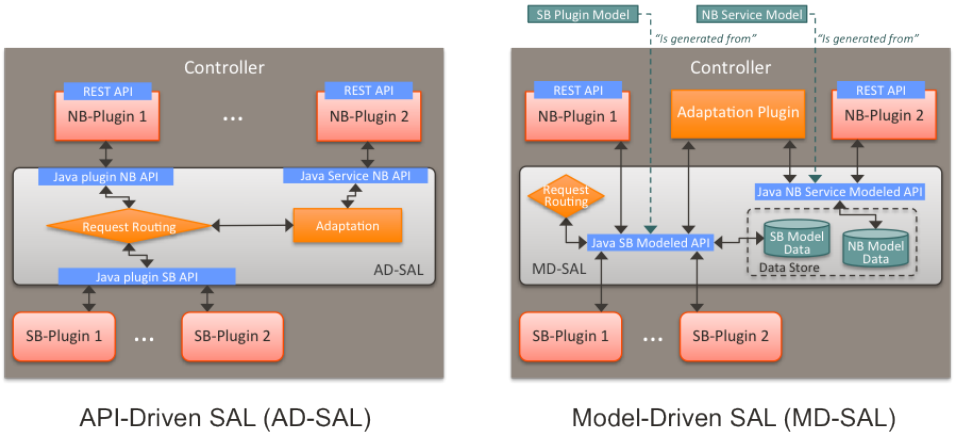


Figura 3.9: Estructuras de las abstracciones AD-SAL y MD-SAL. Extraída de [30]

Básicamente, AD-SAL permite el desarrollo de aplicaciones para el controlador independientemente del protocolo SDN, y por tanto del elemento de red que recibiría las instrucciones de dichas aplicaciones. Fue la primera abstracción creada. Posteriormente, se desarrolla MD-SAL, que unifica todas las APIs, NorthBound y SouthBound, así como las estructuras de datos de los componentes, para trabajar por modelos. Para la descripción de dichas estructuras se introduce el lenguaje de modelado YANG, simplificando así el desarrollo de aplicaciones y componentes para el controlador. [28] De esta

forma, no se realiza una adaptación para cada API como con AD-SAL, sino que todas ellas se adaptan, siendo así la abstracción mayor en MD-SAL, ya que con YANG se definen todos los modelos. Ambas abstracciones se muestran en la imagen 3.9, mientras que la tabla 3.1.3 muestra las diferencias entre AD-SAL y MD-SAL. [29]

Ya que MD-SAL facilita la conexión entre proveedores y consumidores, además de ser la versión mejorada de la abstracción inicial, fue la elegida para la resolución de este trabajo, de forma que la compilación se realice mediante la aplicación de modelos.

AD-SAL	MD-SAL
En las APIs, la solicitud de enrutamiento entre consumidores y proveedores y la adaptación de datos son definidos estáticamente en tiempo de compilación.	En las APIs, las peticiones de enrutado entre consumidores y proveedores se definen mediante modelos, y la adaptación de datos se proporciona mediante los plugins de adaptación "interna".
Tiene tanto NorthBound como SouthBound APIs incluso para funciones/servicios mapeados 1:1 entre plugins NorthBound y SouthBound.	Permite ambos plugins, tanto SouthBound como NorthBound, utilizados en la misma API generada a partir de un modelo. Un plugin pasa a ser un proveedor y el otro un consumidor.
Hay una Representational State Transfer (REST) API "dedicada"para cada plugin SouthBound/NorthBound.	Provee una REST API "común"para acceder a los datos y funciones definidas en modelos.
Provee solicitud de enrutado (selecciona un plugin SouthBound basado en el tipo de servicio) y opcionalmente provee adaptación de servicio, si una NorthBound API es diferente de su correspondiente protocolo API.	Provee solicitud de enrutado y de infraestructura para dar soporte a la adaptación de servicios, pero no provee adaptación de servicios por sí mismo, sino que se apoya en los plugins.
La solicitud de enrutado se basa en el tipo de plugin: SAL sabe qué instancia de nodo es servida por cada plugin, y cuando un plugin NorthBound solicita una operación de un nodo, la solicitud es enviada al plugin apropiado, que envía la petición al nodo pertinente.	La solicitud de enrutado se realiza tanto en el tipo de protocolo como en las instancias del nodo, ya que la instancia de datos se exporta desde el plugin dentro de SAL.
No tiene estados.	Puede almacenar datos de modelos definidos por plugins: proveedores y consumidores pueden pueden intercambiar datos a través del almacenamiento.
Limitado a terminales y modelos de servicio con capacidad de flujo.	Soporta cualquier terminal o modelo de servicio y no sólo a aquellos con capacidad de flujo.
Normalmente proporcionan versiones síncronas y asíncronas del mismo método API.	Solo proporciona APIs asíncronas, pero puede ser usada para ambas estrategias, por lo que aunque fomenta una aproximación asíncrona, no descarta la síncrona.

Tabla 3.3: Comparativa entre AD-SAL y MD-SAL

3.2. Herramientas

Como ya se comentó en la sección 4.2, para desarrollar y ejecutar el sistema se utilizó una máquina virtual con la última versión disponible de Ubuntu instalada. En ella se instaló, además de la distribución Lithium del controlador ODL, la última versión del emulador de redes y algunas herramientas de desarrollo de aplicaciones en Java, lo cual se expone a continuación.

3.2.1. Java

Es un lenguaje de programación orientado a objetos que, gracias a su máquina virtual, puede ser ejecutado en cualquier OS. La compatibilidad por tanto es una de sus mayores ventajas. Además, es uno de los lenguajes de programación estudiados previamente, por lo que se disponía de la preparación necesaria para su utilización en este trabajo.

Se utilizó Eclipse [31] como plataforma de desarrollo, ya que proporciona una integración apropiada de Maven, compilador especificado por ODL para Java, en el que se profundiza en la siguiente sección.

Maven

Maven [32] es una herramienta de software para la construcción y gestión de proyectos Java. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos y el orden de construcción de los elementos, por lo que este archivo se considera la unidad fundamental de trabajo de Maven.

La utilización de esta herramienta se remite al propio controlador, que define Maven como compilador para Java y lo hace indispensable para la correcta integración de los elementos desarrollados

3.2.2. Emulador Mininet

Mininet es un emulador que permite crear redes virtuales [33] [34] junto con todos sus elementos en una sola máquina, facilitando la posterior interacción con dichas redes mediante líneas de comandos. Como beneficio adicional, destaca la posibilidad de desarrollar y ejecutar las redes diseñadas en hardware real. Adicionalmente, se trata de una herramienta de código abierto, por lo que su desarrollo se ha visto respaldado en todo momento por la comunidad, consiguiendo un alto nivel de aceptación y difusión.

No es un simulador de red, por lo que no predice el comportamiento de la red, sino que tiene como intención obtener los mismos datos que se

obtendrían en una red real al ejecutar el mismo código, por lo que considera los eventos imprevistos que interfieren en las dichas redes continuamente y recoge esta aleatoriedad en tiempo real. Este es el motivo principal por el que se utiliza un emulador en lugar de un simulador, ya que al recoger todos los eventos posibles de la red, reproduciendo también los fallos que se pueden encontrar en la misma, resulta una solución fiel a la realidad.

Como emuladores se pueden destacar tanto Mininet como EtsiNet, pero este último es una solución comercial de código privado y, ya que se pretende minimizar el coste del proyecto lo máximo posible, se considera que Mininet es la herramienta más apropiada para el mismo.

Una de las ventajas destacables de Mininet es la facilidad y rapidez para poner en marcha una red, así como su funcionamiento en tiempo real, por lo que resulta interactiva. También es escalable y aplicable a elementos hardware sin necesidad de modificar el código, por lo que su realismo y compatibilidad resultan relevantes a la hora de compararlo con cualquier otra herramienta disponible.

Para el diseño de redes, Mininet dispone de una API escrita en Python que permite diseñar diferentes topologías de red, de forma que sea posible trabajar con diferentes escenarios, lo cual será especialmente relevante en la fase de pruebas y evaluación de la solución diseñada. En la sección 5.3 se presenta la topología básica utilizada en el proceso de implementación de la solución, que utiliza como controlador remoto la solución en sí. Mencionar sin embargo que aunque Mininet ofrece multitud de topologías para emular las redes, tanto por defecto como programables con Python, el esquema base viene dado por el comando de la figura 3.10, que da lugar a la estructura mostrada en la misma.

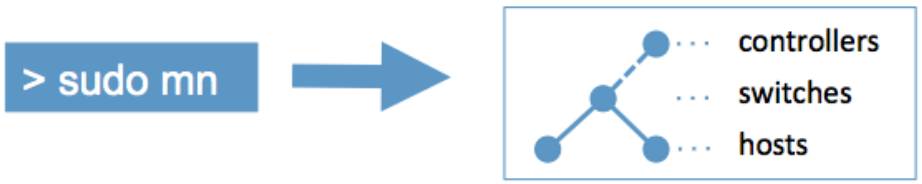


Figura 3.10: Generación y esquema de la estructura básica en Mininet

3.3. Protocolo de Internet (IP)

El protocolo de Internet IP es un protocolo de comunicación de datos digitales utilizado en la capa 3 o capa de red del modelo Open System Interconnection (OSI). Su función principal es el envío de paquetes de datos, tanto a nivel local como a través de redes. IP trata de realizar la entrega de

dichos paquetes de datos mediante técnicas de encaminamiento de la mejor forma posible, sin garantizar que se alcance el destino final, pero buscando la mejor ruta posible de entre las conocidas por la máquina que esté haciendo uso del protocolo. [35]

Actualmente, se encuentra en uso IPv4, aunque se pretende que en un futuro próximo sea sustituido por su versión mejorada, IPv6. En principio, este proyecto se plantea como una implementación sobre IPv6, ya que las vías futuras de investigación se enfocan hacia la implantación de esta versión del protocolo. Las razones de este enfoque se plantean a continuación, en la sección 3.3.2; sin embargo, tras la implementación del controlador para IPv6 se realizó un filtrado del tráfico para IPv4, demostrándose así la escalabilidad de la solución, por lo que también se ha realizado una revisión del estado del arte de esta versión.

3.3.1. IPv4

IPv4 es la cuarta versión del protocolo de Internet y la primera en ser implementada a gran escala. Utiliza direcciones de 32 bits, en su mayoría dedicadas a Local Area Network (LAN), en español redes de área local. Como se ha comentado en 2.3, debido al crecimiento de Internet y al desperdicio de direcciones, IPv4 limita el uso de Internet en sí, lo que ha motivado la implantación de IPv6 como sustituto de esta versión.

3.3.2. IPv6

La versión 6 del protocolo de Internet surge con el objetivo de reemplazar a su antecedente IPv4, ya que el número de direcciones de red admisibles por éste restringe tanto el crecimiento de Internet como su uso en sí. Así, especifica un nuevo formato de paquete, minimizando el procesamiento del encabezado. Algunos de los cambios más relevantes que se plantean sobre IPv4 se presentan a continuación. [12]

- **Capacidad extendida de direccionamiento.** En esta versión, es posible reasignar la numeración de toda la red cambiando únicamente el prefijo anunciado por unos pocos routers, ya que los identificadores de los nodos pueden ser configurados automáticamente independientemente por cada nodo. De esta forma, aunque la tasa de utilización del espacio de direcciones sea menor, tanto la administración de redes como el "ruteo"serán más eficientes.
- **Autoconfiguración de direcciones libres de estado, SLAAC.** Gracias a Internet Control Message Protocol (ICMP)v6, los nodos pueden configurarse a sí mismos automáticamente al conectarse a una

red. [37] Así, la primera vez que un nodo se conecte a una red ruteada en IPv6, enviará una solicitud de router (Router Solicitation) de enlace local mediante multicast, pidiendo los parámetros de configuración de red. El router, de estar preparado, responderá con un anuncio de router (Router Advertisement). Se realizará también la solicitud (Neighbor Solicitation) y aviso (Neighbor Advertisement) de nodos correspondiente para comunicarse con el resto de elementos de la red. Todo este procedimiento se conoce como descubrimiento de vecino, y se analizará en profundidad en el capítulo 5, así como la estructura de las tramas de IPv6, estudiada para el diseño de la solución. Si la autoconfiguración de las SLAAC no es adecuada para una aplicación, existe la posibilidad de utilizar DHCPv6, así como de configurar los nodos de forma estática. Respecto a la configuración de direcciones de los routers, se trata de un caso especial, ya que al ser fuente de configuraciones tiene unos requerimientos específicos.

- **Multicast.** Es la habilidad de enviar un paquete único a destinos múltiples y forma parte de la especificación base de IPv6. En lugar de broadcast, en este caso se envía un paquete al grupo de multicast de enlace local todos los nodos (all hosts), por lo que no existe una dirección de broadcast y la dirección más alta de la red, que equivaldría a la de broadcast en IPv4, es una dirección de IPv6 también.

- **Procesamiento simplificado de los routers.** Se simplifica la cabecera de los paquetes, así como el proceso de reenvío de los mismos, de forma que el procesamiento por parte de los routers sea más simple y eficiente. Los campos poco utilizados del encabezado se sitúan en opciones separadas, por lo que aunque las direcciones son cuatro veces más largas, el encabezado será tan sólo el doble que el de IPv4, sin tener en cuetna esas opciones. Además, los routers no realizarán la fragmentación, el encabezado no incluye checksum (lo cual es relevante para los routers por software, ya que no tendrán que calcularlo) y el campo TTL para a ser el límite de saltos permitidos entre routers.

- **Direccionamiento.** El cambio más significativo entre ambas versiones es la longitud de las direcciones de red. En este caso, serán de 128 bits, lo que equivale a 32 dígitos hexadecimales, y normalmente se dividen en dos partes lógicas de 64 bits cada una: un prefijo y un identificador de interfaz, que será generado automáticamente a partir de la dirección física, Media Access Control (MAC), de la interfaz a la que está asignada esa dirección.

3.4. Tormentas de Broadcast

En [3], Cisco realizó una serie de pruebas para medir el efecto de broadcast en una estación Sun SPARC 2 con una tarjeta estándar de Ethernet. La SPARCstation disponía de un sistema operativo SunOS versión 4.1.3 con IP multicast desactivado. Se demostró que una estación de trabajo puede dejar de funcionar debido a las inundaciones broadcast de la red. Además, se observaron puntualmente picos de miles de broadcast por segundo en las tormentas de broadcast, lo que tal y como muestra la figura 3.11 supone una disminución del rendimiento del sistema de hasta el 25 %.

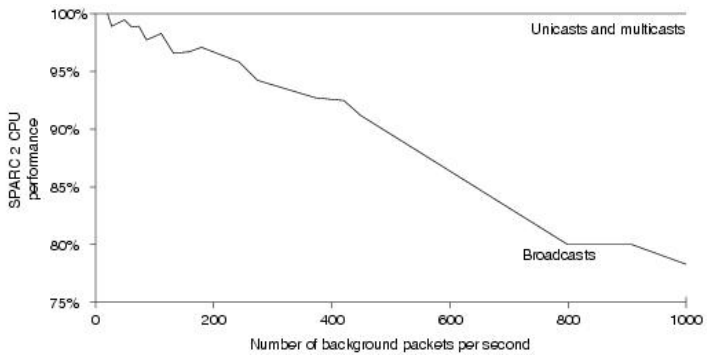


Figura 3.11: Efecto de una tormenta de broadcast en hosts de redes IP. Extraída de [3]

Broadcast, también conocido como difusión amplia, es básicamente una forma de transmisión en la que un nodo emisor envía información a una multitud de nodos receptores de manera simultánea. Una tormenta de *broadcast* implica la saturación de la red debido a la emisión constante de paquetes o al tráfico multidifusión. Transmitir tormentas puede concluir con una pérdida completa de la conectividad de la red, así como de los paquetes que se van a enviar, ya que el reenvío de un gran número de paquetes por parte de un puerto supone una reducción del rendimiento de la red y podría incluso interrumpir el servicio.

La escalabilidad de las redes Ethernet está limitada principalmente por las inundaciones de red causadas por los protocolos de arranque (bootstrapping protocols) de los usuarios finales, como ARP, DHCP, etc. Para superar estas limitaciones, surgen nuevas arquitecturas basadas en Ethernet, como Ethane [38] y SEATTLE [39], que pretenden reducir las inundaciones de tramas de broadcast sin dejar de ofrecer los servicios esperados de Ethernet.

SEATTLE tiene como objetivo proporcionar un protocolo sin configuración que resulte escalable a redes más extensas. Para ello, implementa un directorio a nivel de red con Distributed Hash Tables (DHT) en los switches

propios de la arquitectura para conservar la ubicación de la dirección MAC de cada host. Adicionalmente, utiliza un protocolo de anuncio de estado unicast de enlace para evitar broadcast en la actualización de las tablas. Los mensajes ARP y DHCP de broadcast se convierten en peticiones unicast al servicio de directorio. Sin embargo, aunque SEATTLE se plantea como una soluciónn prometedora para escalar redes Ethernet, requiere unos switches no estándar muy costosos.

Ethane es otra arquitectura con una filosofía similar a OpenFlow. Aunque Ethane se dirige a la definición y al refuerzo de políticas en toda la red, aborda el problema del broadcast delegando en el controlador el tratamiento de protocolos de arranque como ARP, reduciendo de esta forma el tráfico de broadcast y el tamaño de las tablas MAC de los switches.

Capítulo 4

Recursos y planificación

Este capítulo describe la planificación temporal que ha supuesto el desarrollo de este proyecto, así como la estimación de costes que supondría su realización. Se tienen en cuenta por tanto todas las fases del trabajo, desde la revisión del estado del arte hasta el análisis final de los resultados obtenidos, así como los recursos utilizados, con la finalidad de realizar una estimación del presupuesto final que sería necesario para cubrir la realización completa del proyecto.

4.1. Planificación temporal

En esta sección se presenta la planificación temporal. Para una mejor comprensión por parte del lector, se ha dividido la carga de trabajo realizado en diferentes actividades, tal y como muestra la tabla 4.1. Este desarrollo se ha representado gráficamente con un diagrama de Gantt, presente en la figura 4.1, de forma que las diferentes fases del desarrollo queden ilustradas acompañando a la tabla.

La idea de esa sección es acercar el desarrollo del proyecto a la realidad lo máximo posible, teniendo en cuenta todos los factores que intervienen en el mismo de forma precisa y detallada.

Las actividades en las que se ha dividido la carga de trabajo, por tanto, se describen a continuación:

- **Revisión del estado del arte.** En este primer paquete de trabajo se recabó información para la puesta en marcha del proyecto. Se realizó una revisión bibliográfica, junto con una recopilación de datos y conceptos necesarios para la comprensión del trabajo y para el desarrollo del mismo, profundizando en las tecnologías implicadas, especialmente SDN y ODL.

- **Familiarización con las herramientas.** Esta fase incluye tanto la configuración del sistema como la toma de contacto con el emulador de redes utilizado: Mininet. Debido a diferentes problemas de compatibilidad encontrados a lo largo de dicha configuración, tal y como se detalla en el apéndice B, se optó por la utilización de una máquina virtual, en la que se realizó la instalación completa de Mininet. También fue necesario, en el proceso de acercamiento a la herramienta, aprender conceptos básicos de Python, ya que es el lenguaje utilizado para crear diferentes topologías de red.

- **Familiarización con el controlador.** Es la primera actividad incluida en lo que se considera el desarrollo de la solución en sí. Una vez realizada la revisión del estado del arte, se determinó que el controlador remoto que mejor se adecuaba a las necesidades del proyecto sería ODL, por lo que se realizó un estudio exhaustivo del mismo mediante otra máquina virtual preparada a modo de tutorial, con el fin de comprender y manejar apropiadamente este elemento de la red. Junto con ODL, fue necesario estudiar Java, así como Maven, herramienta utilizada en gestión y construcción de proyectos Java, con la que se realizó la compilación de la solución.

- **Familiarización con los protocolos.** Antes de comenzar el diseño de la solución, es necesario analizar los diferentes protocolos implicados, así como determinar las posibilidades que plantean, que se aplicarán posteriormente en el diseño del filtro del sistema. Particularmente, se tienen en cuenta los protocolos de Internet IPv4 e IPv6, junto con los mensajes que cada uno supone en el red. Para ello se utiliza tanto el modo depuración del controlador como el analizador de protocolos Wireshark, que serán agentes imprescindibles a partir de esta fase en el resto de actividades de diseño, implementación y evaluación de la solución.

- **Diseño de la solución.** En este paquete de trabajo se parte del esqueleto del "Learning Switch"proporcionado con el entorno de trabajo del controlador para analizar el funcionamiento del mismo y a partir de los métodos ya implementados, diseñar una posible solución para nuestro sistema que aproveche el código disponible.

- **Implementación de la solución.** En esta fase se realiza la implementación en sí de la solución, así como la resolución de los posibles problemas de diseño y rendimiento. Incluye la implementación del controlador completa, teniendo en cuenta las diferentes fases del proceso, a saber detección, filtrado y reenvío de paquetes, para los distintos protocolos estudiados, IPv4 e IPv6. Requiere por tanto la configuración de diferentes escenarios en Mininet, así como del protocolo IPv6

para la asignación de direcciones en los hosts de dichos escenarios, lo cual queda documentado en el apéndice A.

- **Evaluación de la solución propuesta.** Fase de pruebas y de análisis de los resultados obtenidos. Una vez implementada la solución, se realizan una serie de pruebas para comprobar su rendimiento en diferentes topologías. Se examina así su correcto funcionamiento con el fin de detectar posibles fallos no contemplados en el periodo de implementación, así como de recoger datos acerca del comportamiento del sistema. Concretamente, se estudia la carga de paquetes enviados y procesados con el controlador sin modificar y con el implementado, así como las consecuencias de esa diferencia de carga de la red que supone la solución.

- **Elaboración de la memoria.** Este último paquete incluye el proceso de elaboración del documento técnico, compuesto por la fase de preparación, en la que se realiza una familiarización con el lenguaje Látex y la instalación de los paquetes necesarios para utilizarlo, y de redacción, con la que se pretende documentar todos los aspectos teóricos y prácticos del trabajo realizado.

Paquetes de trabajo	Fecha de inicio	Fecha final prevista	Duración en días de trabajo
Trabajo Fin de Grado	01.09.2015	30.06.2016	303
Revisión del estado del arte	01.09.2015	31.10.2015	60
Familiarización con las herramientas	15.10.2015	31.12.2015	77
Desarrollo de la solución	15.12.2015	15.05.2016	152
Familiarización con el controlador	15.12.2015	31.01.2016	47
Familiarización con los protocolos	01.02.2016	28.02.2016	27
Diseño de la solución	15.02.2016	31.03.2016	45
Implementación de la solución	01.04.2016	15.05.2016	44
Evaluación de la solución propuesta	15.05.2016	15.06.2016	31
Elaboración de la memoria	01.05.2016	30.06.2016	60

Tabla 4.1: Planificación temporal del proyecto

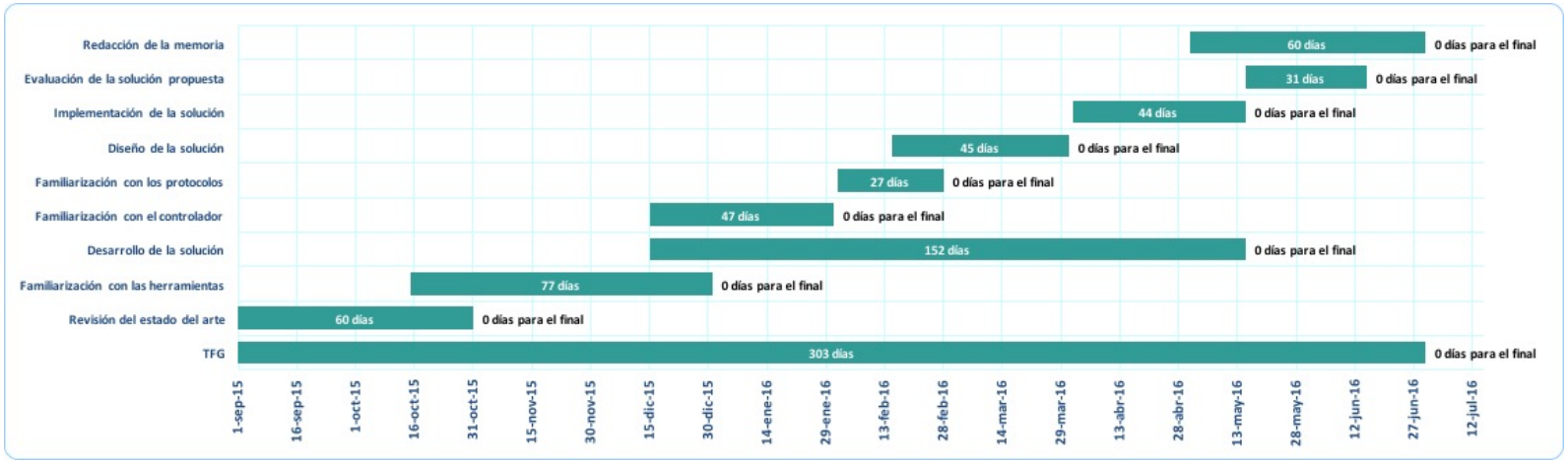


Figura 4.1: Diagrama de Gantt de la planificación temporal

Por último, a modo de conclusión, se estima, considerando los días trabajados, en 500 horas el total de horas empleadas en la realización del proyecto. Esta estimación se aplicará en el apartado 4.3 con el fin de evaluar los costes asociados a los recursos humanos que ha requerido este trabajo.

4.2. Recursos utilizados

A continuación se desglosan los recursos implicados en la realización del proyecto, diferenciando cuando se trata de recursos humanos, Software o Hardware.

4.2.1. Recursos humanos

- D. Juan José Ramos Muñoz, Profesor del Área de Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, en calidad de tutor del proyecto.
- D. Jorge Navarro Ortiz, Profesor del Área de Comunicaciones del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, en calidad de cotutor del proyecto.
- Bárbara Valera Muros, alumna del Grado en Ingeniería en Tecnologías de Telecomunicación, en calidad de investigadora y autora del proyecto.

4.2.2. Recursos Software

- Oracle VirtualBox, máquina virtual sobre la que instala el sistema operativo con las herramientas necesarias para trabajar con SDN.
- Sistema Operativo Ubuntu 14.04 (64 bits), ya que es el que mejor se adecua al entorno de desarrollo y mejor compatibilidad ofrece para SDN.
- Mininet, herramienta esencial en la emulación de redes SDN.
- OpenDayLight, controlador seleccionado para operar con SDN y sobre el que se realizará la implementación de la solución.
- Java Development Kit, JDK.
- Maven, herramienta necesaria para la compilación y manejo de paquetes del proyecto Java.

- Eclipse Luna, entorno de desarrollo integrado, Integrated Development Environment (IDE), utilizado para la programación del controlador.
- Wireshark, analizador de protocolos utilizado para capturar los paquetes procesados por el controlador.
- Texmaker, programa de edición en Látex para la redacción del documento técnico, junto con un compilador Látex compatible con el sistema operativo del equipo utilizado, OS X.
- Overleaf, sistema de escritura colaborativo, para facilitar el seguimiento del proyecto por los tutores del mismo.

4.2.3. Recursos Hardware

- Equipo personal con procesador Intel Core i7 a 2.7GHz, memoria RAM de 4GB y unidad de estado sólido, Solid-State Drive (SSD) de 500GB de capacidad. Utilizado para la implementación del controlador, configuración de los diferentes escenarios y evaluación del sistema en la fase de pruebas.
- Línea de acceso a Internet. En la fase de revisión del estado del arte, acompañada de permisos de acceso a portales de artículos académicos, que gracias a un convenio universitario fueron gratuitos.

4.3. Estimación de costes

Una vez identificados los recursos involucrados en el proyecto, se realiza una estimación del coste que supondría abordar este trabajo.

En primer lugar, respecto al Software utilizado. En un intento de minimizar el coste del proyecto, se ha recurrido al uso del llamado Software Libre, de forma que las herramientas sean gratuitas y no supongan coste alguno. Además, tal y como se menciona en el apartado 4.2.3, gracias al convenio universitario el coste asociado al acceso a artículos académicos es también nulo.

Por otra parte, el coste asociado al Hardware y los recursos humanos implicados se detalla a continuación, en el apartado 4.3.1.

4.3.1. Presupuesto final

Se considera que el proyecto tuvo una duración de 10 meses, tal y como se estimaba en 4.1, dato a tener en cuenta en el cálculo del coste de la línea de acceso a Internet, que supone 25€/mes.

Respecto a los recursos humanos implicados, se establece que el sueldo medio de un alumno recién titulado es de 25€/hora, ya que se estima que los egresados con menos de 5 años de experiencia ingresan una media de 25.000€ netos/año, mientras que el del profesor contratado será de 50€/hora. Tal y como se menciona en 4.1, las horas trabajadas por el alumno son 500. Como este trabajo fue dirigido por dos tutores, se estima que el tutor principal trabajó 40 horas en la tutorización y 10 en la revisión, mientras que el cotutor 10 horas, principalmente en tutorización.

En la tabla 4.3.1, se muestra el coste asociado a cada recurso, junto con una estimación del presupuesto total que supondría abordar este proyecto, mientras que en los gráficos 4.2 y 4.3 se muestran unas representaciones del porcentaje de costes asociado a cada recurso.

Recurso	Coste asociado	Coste total
Trabajo del alumno	25 €/hora	12500 €
Tutorización Juan José Ramos	50 €/hora	2500 €
Tutorización Jorge Navarro	50 €/hora	500 €
Línea de acceso a Internet	25 €/mes	250 €
Equipo personal de trabajo	1300 €	1300 €
Software empleado	0 €	0 €
PRESUPUESTO TOTAL		**17050 €**

Tabla 4.2: Presupuesto total estimado

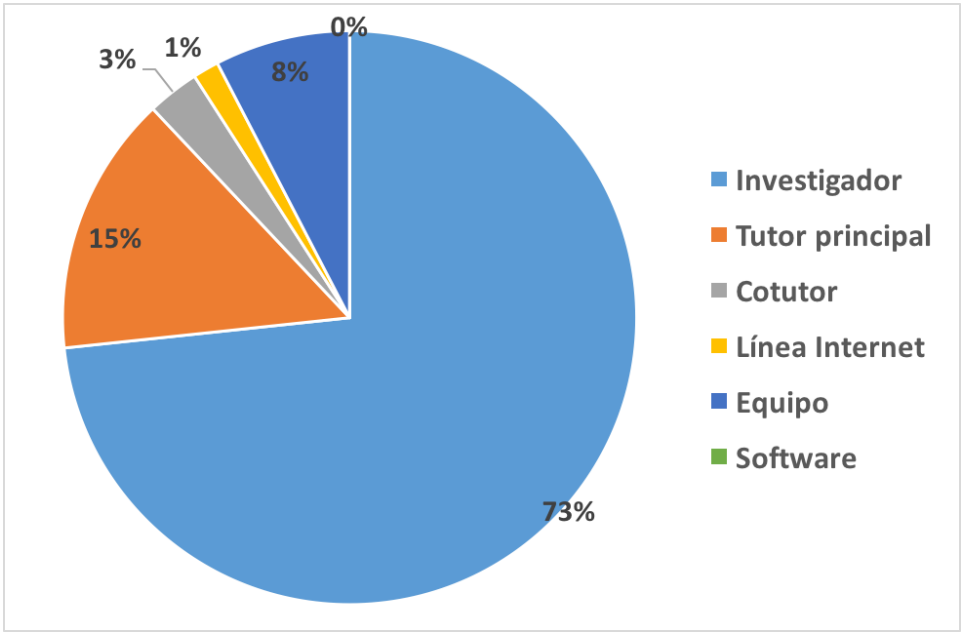


Figura 4.2: Porcentaje final de costes asociados a cada recurso

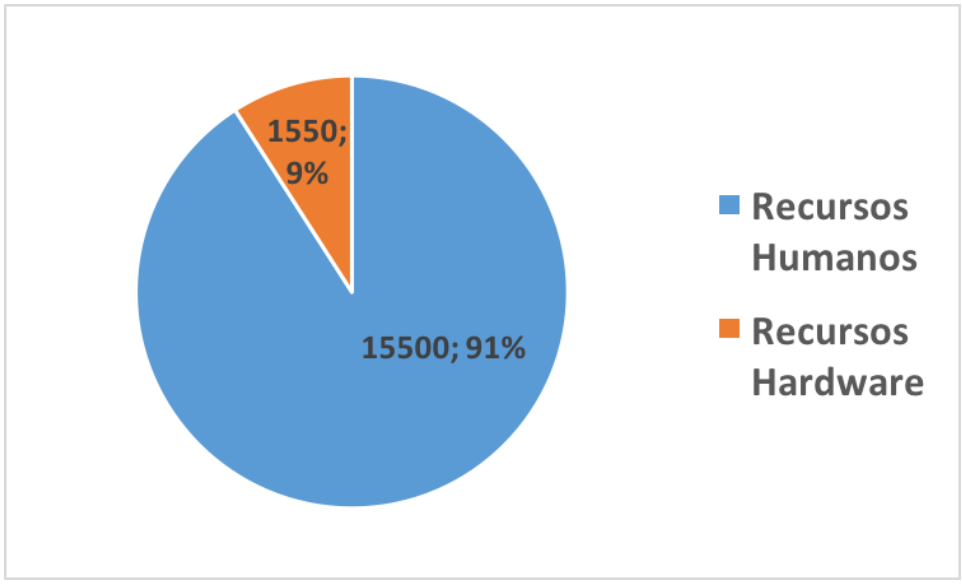


Figura 4.3: Porcentaje final de costes según el tipo de recurso

Capítulo 5

Resolución del trabajo

Este capítulo se centra en el desarrollo del trabajo en sí, planteando la problemática inicial sobre la que se basa el diseño de la solución y analizando dicho diseño. Posteriormente, se presenta la implementación de cada una de las partes en las que se ha dividido la solución, así como la justificación de cada uno de los métodos implementados.

5.1. Problemática inicial

Tal y como plantea la sección 3.4, el rendimiento de un sistema se ve gravemente afectado por las inundaciones que los mensajes broadcast causan en la red. Dichos mensajes se asocian a los protocolos de arranque de los usuarios finales y limitan la escalabilidad de las redes Ethernet.

Dado que el objetivo principal de este trabajo es implementar una red basada sobre Ethernet, se pretende realizar un filtrado de esos mensajes de broadcast mediante un controlador desarrollado sobre SDN, diseñado inicialmente para actuar en IPv6, aunque posteriormente se escala al protocolo ARP de IPv4, que se presenta como uno de los protocolos que más tráfico broadcast genera.

Se tiene también en cuenta la posibilidad de escalarlo en un futuro a otros protocolos, por lo que se realiza un diseño que permita dicha adaptación sin la necesidad de una programación excesiva, sino reutilizando los métodos ya implementados.

5.1.1. ARP

El protocolo ARP es un protocolo de comunicaciones de la capa de red encargado de encontrar la dirección MAC que corresponde a una determina-

da dirección IP. [8] En Ethernet, la capa de enlace trabaja con direcciones físicas, por lo que hará uso de este protocolo para traducir las direcciones mediante las tablas ARP en las que se almacenan.

Floodless Service Discovery Model (FSDM) se presenta como posible solución al problema de la escalabilidad en redes Ethernet introducido por las tormentas de broadcast que causan protocolos como ARP. Para escalar Ethernet a redes con millones de nodos, es necesario mantener una gran cantidad de entradas, lo que afecta al sistema tal y como muestra la imagen 5.1, lo que de nuevo plantea el problema del rendimiento de las redes frente al broadcast, que se pretende solucionar en este proyecto. [2]

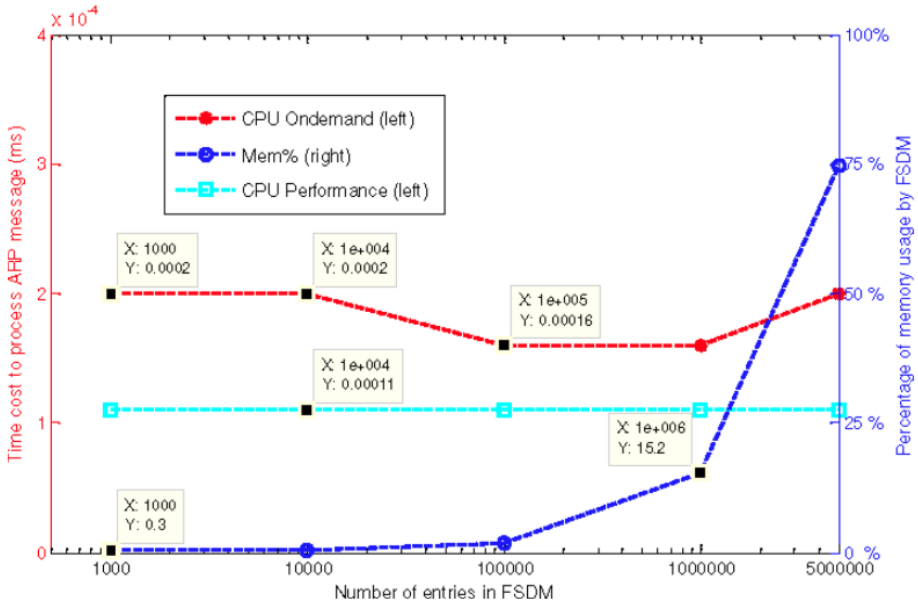


Figura 5.1: Tiempo necesario para procesar un mensaje ARP y porcentaje de memoria en uso con FSDM. Extraída de [2]

ARP es por tanto uno de los límites para la escalabilidad en redes Ethernet. Tal y como muestra la tabla 5.1.1 el tráfico ARP constituía en 2009 el 87.4 %, 88.2 % y 85 % de todo el tráfico broadcast producido por Rice, Yahoo! y Emulab. [36] Todo esto justifica la implementación de una solución que filtre los mensajes generados por este protocolo con la intención de mejorar el rendimiento de la red.

	ARP	IPX/ SAP	Net-BIOS	SUN-RPC	DHCP	IPP	MS-SQL	X11	XD-MCP	OSI	ISAKMP
Rice	235735	405	19560	9590	1272	3014	4	8	4	122	0
Yahoo!	986248	0	237	0	0	0	0	0	0	0	131672
Emulab	110586	0	328	0	19154	0	0	0	0	0	0

Tabla 5.1: Desglose del número de paquetes broadcast medidos en diferentes redes Ethernet durante 24h en un día laborable.

5.2. Diseño de la solución

En primer lugar, se procede al diseño de la solución en sí de forma abstracta, sin tener en cuenta las características específicas de cada protocolo, sino lo que debe hacer el controlador en cada una de las fases.

Se divide el proceso en tres fases bien diferenciadas, que componen tanto el diseño como la implementación:

a. Detección de paquetes.

b. Filtrado de paquetes.

c. Reenvío de paquetes.

Así, tal y como muestra el diagrama 5.2, el host origen, que inicia la solicitud para cualquiera de los protocolos a implementar, mandará un mensaje que es procesado por el controlador, a partir de lo cual surgen dos posibles rutas:

1. Si el controlador no dispone de información suficiente para procesar la solicitud, la reenvía al destino, que generará una respuesta. Dicha respuesta es de nuevo procesada por el controlador, que almacenará los datos para resolver futuras peticiones, y reenviada al origen de la solicitud. Esto se muestra en el diagrama de color naranja.

2. Si el controlador dispone de información suficiente para atender la solicitud, envía directamente una respuesta al host origen, sin necesidad de implicar al host destino en este proceso. Este camino se representa de color azul.

De esta forma, aunque inicialmente la mayoría de mensajes sigan el primer procedimiento, una vez almacenados los datos de los host que forman parte de la red el controlador dispondrá de datos suficientes como para atender las solicitudes de los nuevos hosts sin necesidad de implicar al resto, lo que reducirá el tráfico de la red, incrementando su rendimiento.

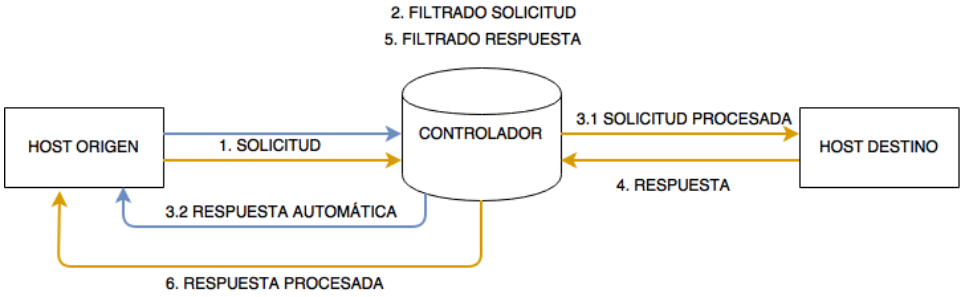


Figura 5.2: Esquema básico de la comunicación establecida entre los hosts y el controlador.

Esto se observa claramente en el gráfico 5.3, en el que se puede apreciar que el primer host invierte más tiempo esperando respuesta cuando el controlador no tiene la información almacenada y hay que seguir la primera ruta, en naranja.

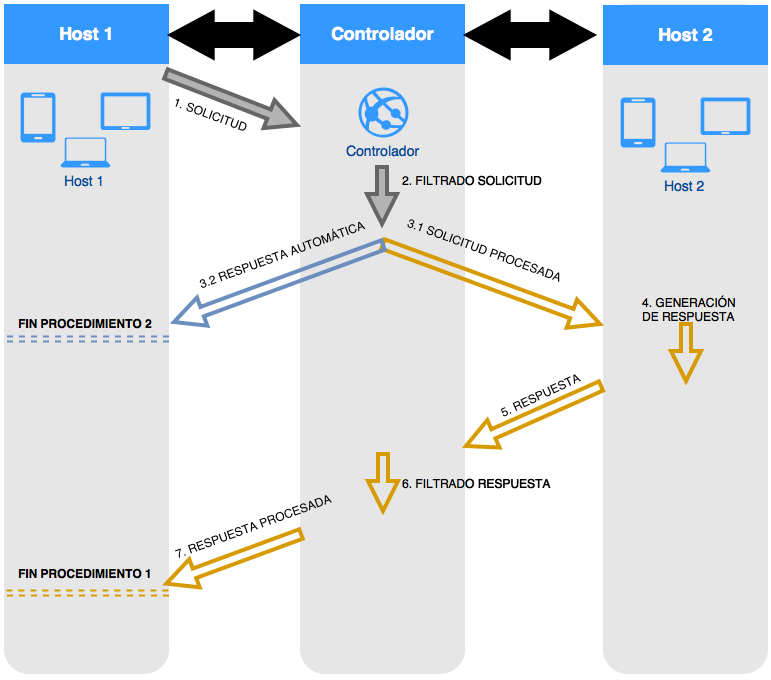


Figura 5.3: Diagrama de secuencias de la comunicación entre hosts y controlador.

5.2.1. Detección de paquetes

En primer lugar, se realizará la detección e identificación de paquetes para su posterior clasificación y filtrado. Tal y como muestra el diagrama 5.4, se analiza la llegada de tres tipos de paquetes, según el tipo de protocolo:

- **IPv6.** Con código Ethernet 0x86DD, cuando el controlador detecte un paquete IPv6 analizará si se trata de un mensaje ICMP. De no ser así, el proceso habrá terminado y reenviará el mensaje sin intervenir. De ser ICMP, extraerá los bytes del *payload* correspondientes a las direcciones MAC e IP de origen y destino y continuará con el proceso de filtrado, dando por terminada la detección.

- **IPv4.** Con código Ethernet 0x0800, el controlador extraerá las direcciones IP y MAC y comprobará si existe alguna entrada en la tabla para dichas direcciones. De ser así, el proceso habrá acabado. En caso contrario, creará la entrada, finalizando así el proceso de detección para IPv4, para el que no existe filtrado, por lo que se realizará un reenvío como si el controlador no hubiese intervenido.

- **ARP.** Con código Ethernet 0x0806, el controlador extrae las direcciones MAC e IP de origen y destino y continúa con el proceso de filtrado y reenvío, sin realizar ninguna otra comprobación.

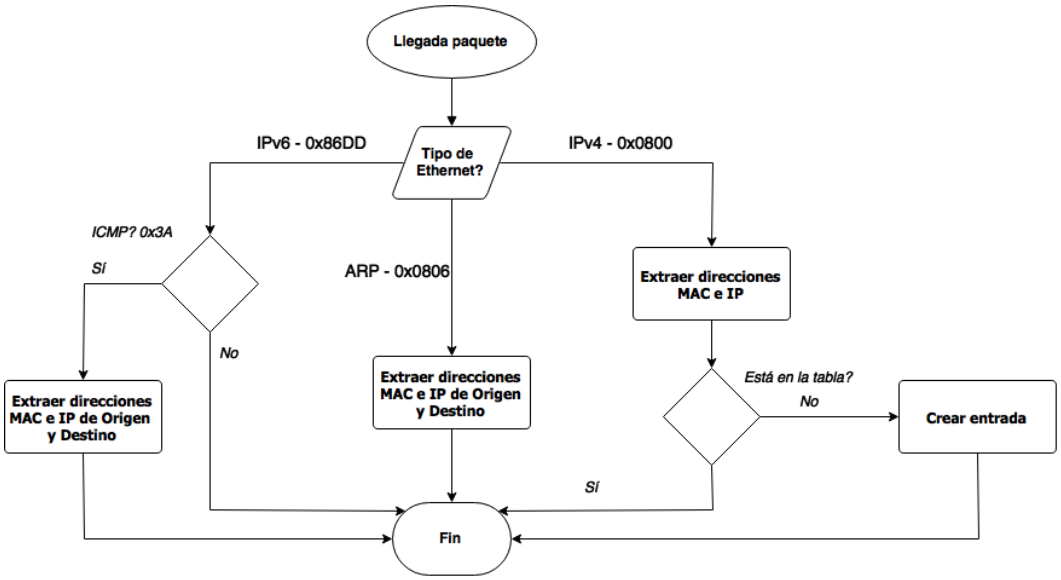


Figura 5.4: Diagrama de flujo del proceso de detección de paquetes.

5.2.2. Filtrado de paquetes

Como se ha mencionado en la sección anterior, tras la detección de paquetes se realiza el filtrado, para el caso de IPv6 y ARP. Con este filtrado se determinará los paquetes que deben ser reenviados y a los que responderá el controlador directamente.

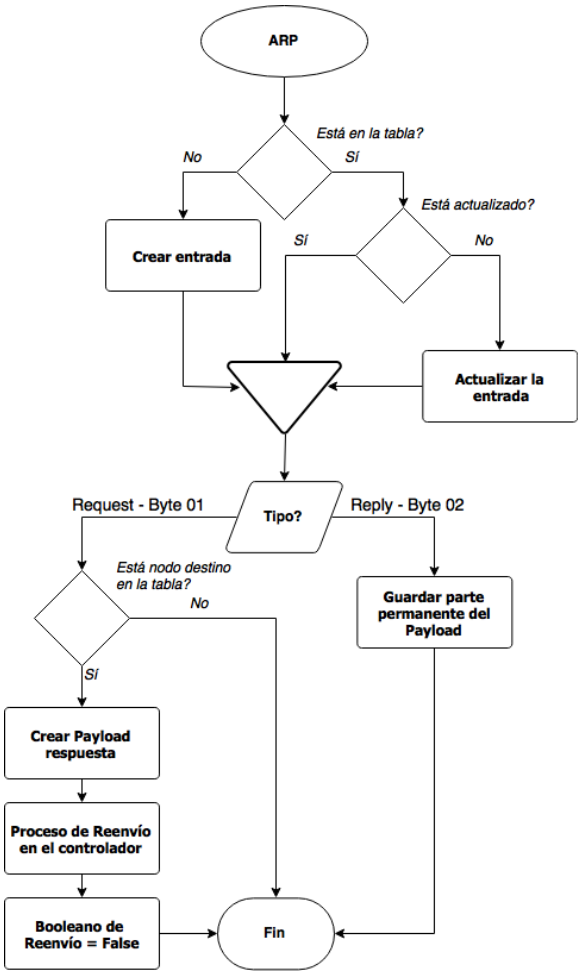


Figura 5.5: Diagrama de flujo del proceso de filtrado para ARP.

- **Caso ARP.** En primer lugar, tras haber realizado la extracción de las direcciones del origen, se comprobará si dicho host se corresponde con alguna de las entradas de las que dispone el controlador en su tabla. De no ser así, se crea la entrada; en caso contrario, se comprueba si la entrada está actualizada, ya que en ARP cada entrada tiene una duración de 2 minutos, y de ser necesario se actualizará. Tras esto, se estudiará qué tipo de mensaje es, tal y como muestra el diagrama 5.5:

01 **Request.** Comprueba si el nodo destino corresponde con alguna entrada de la tabla. Si existe esa entrada, el controlador genera un *payload* respuesta automáticamente, que envía al nodo origen sin necesidad de reenviar la petición al nodo destino, por lo que anula el proceso de reenvío de esa *request*. En caso de no existir la entrada, pasará al proceso de reenvío directamente, planteado en la sección 5.2.3.

02 **Reply.** De haberse generado un mensaje *reply* se reenvía con normalidad, almacenando la parte permanente del *payload* de este tipo de mensajes para posteriormente construir las respuestas automáticas cuando el controlador recibe una petición. Se entiende que estos mensajes se generan únicamente como respuesta a peticiones que no pueden ser atendidas directamente por el controlador, lo que justifica que no se plantee la posibilidad de cancelar el reenvío de este paquete.

- **Caso IPv6.** En este caso, dado que hay más posibilidades [40] dentro de los casos de ICMP, se presenta un diagrama mucho más complejo, tal y como se observa en 5.6.

0x85 **Router Solicitation.** Se genera al añadirse una nueva interfaz a la red. Los nodos que forman parte de dicha interfaz solicitan al router un anuncio con el prefijo de red que les permita configurarse para habilitar la comunicación con el resto de nodos que forman parte de la red. Por tanto, habrá dos posibilidades al generarse este mensaje:

  - El router ha sido también añadido a la red en ese instante, por lo que el controlador no dispone de la información necesaria para responder al nodo solicitante y permitirá el mensaje continuar hacia el resto de la red, realizando un reenvío normal del paquete.
  - El router había enviado previamente un anuncio a la red, que había sido recibido y almacenado por el controlador, por lo que éste mismo responde directamente con el prefijo de configuración al nodo solicitante, bloqueando el reenvío de la solicitud de router.

0x86 **Router Advertisement.** Se genera tanto como respuesta a un Router Solicitation como periódicamente. En este caso, de no existir una entrada previa de router en la red, se creará, almacenando también su prefijo de red y el payload, necesario para responder a las solicitudes posteriores. En caso de existir la entrada, se comprobará si está actualizada y de ser necesario, se actualizará. De estar actualizada, se bloquea el reenvío de este

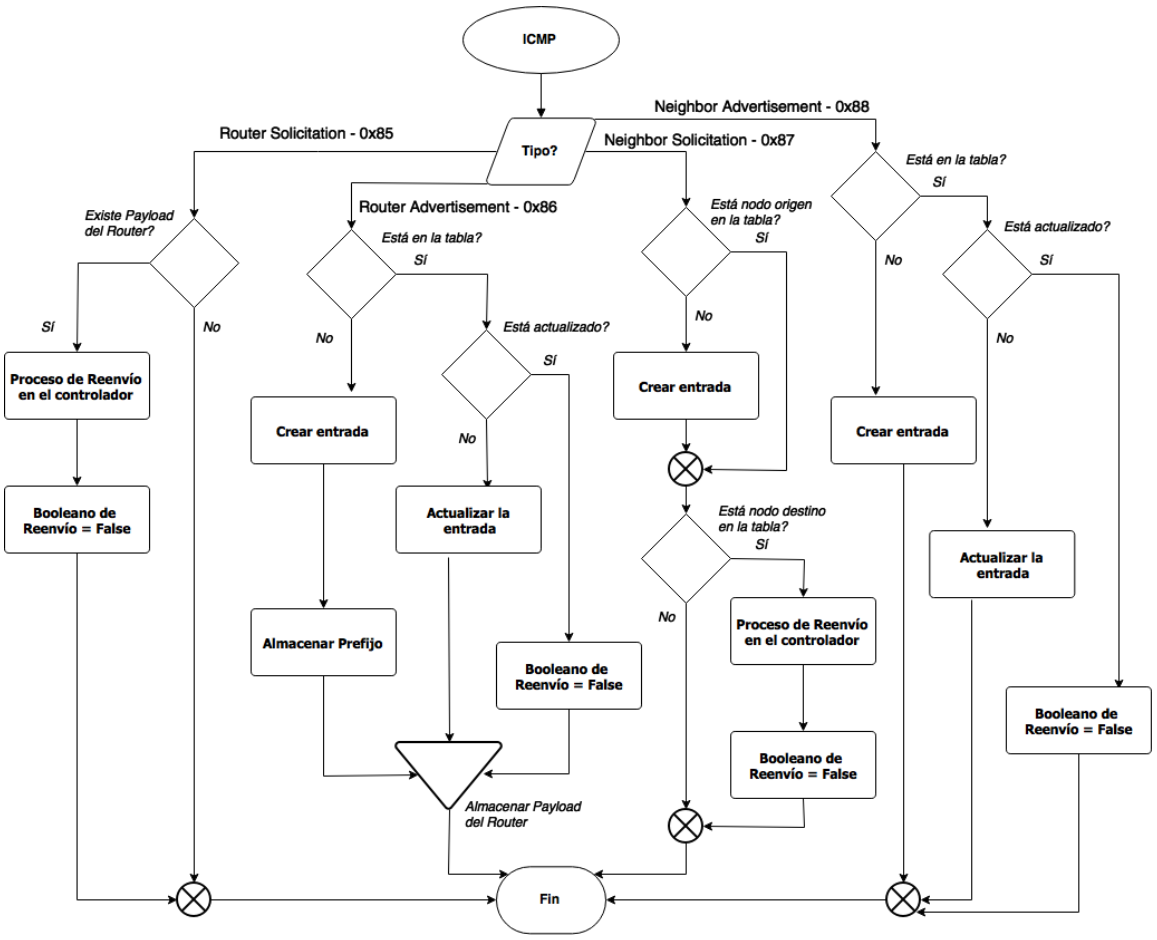


Figura 5.6: Diagrama de flujo del proceso de filtrado para IPv6.

paquete al resto de la red, ya que en caso necesario, como lo es responder a una solicitud, se encargará de responder el controlador automáticamente.

0x87 **Neighbor Solicitation.** Generado tanto para conocer la dirección de un nodo de la red con la que el solicitante se quiere comunicar, como para realizar la autoasignación de direcciones IP evitando las direcciones duplicadas. De no estar el solicitante en la tabla del controlador, o disponer el controlador de una información desactualizada, se crea una entrada para dicho nodo. Posteriormente, de disponer el controlador de información suficiente sobre el destino para responder directamente, generará dicha respuesta y bloqueará el reenvío de la solicitud a la red. En caso contrario, permitirá el reenvío de la solicitud, a la espera de un anuncio de vecino.

0x88 **Neighbor Advertisement.** Generado tanto como respuesta a la solicitud, en cuyo caso se crea la entrada correspondiente a este nodo (de existir dicha entrada se habría bloqueado y respondido la solicitud automáticamente) y se continúa con el reenvío, como para anunciar la llegada de un nodo a la red, en cuyo caso se comprueba si dicho nodo había estado en la red con anterioridad y disponía de una entrada desactualizada. De haber estado anteriormente y disponer de una entrada actualizada, se bloquea el anuncio para agilizar el tráfico de la red.

5.2.3. Reenvío de paquetes

Por último, se realiza la fase de reenvío de paquetes, de acuerdo con el resultado obtenido en el proceso de filtrado. Así, tal y como muestra el diagrama 5.7, se define una variable de reenvío que, de no modificarse, realizará el envío del paquete a su destinatario inicial. En caso contrario, el propio controlador atiende la petición recibida, cancelando el proceso de reenvío de ese paquete y sustituyéndolo por el envío de una respuesta generada por el mismo controlador y dirigida al nodo origen de la petición.

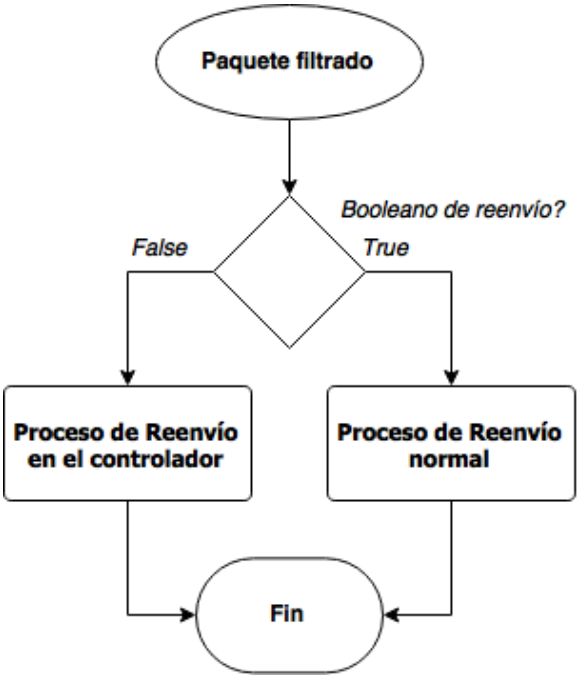


Figura 5.7: Diagrama de flujo del proceso de reenvío de paquetes.

5.3. Implementación de la solución

A continuación, se presenta el proceso de implementación. Al igual que en el apartado anterior y con la intención de facilitar la comprensión de los métodos implementados, se divide la solución en tres fases: detección, filtrado y reenvío de paquetes. Previamente se expone el escenario sobre el que se ha programado el controlador.

5.3.1. Entorno de desarrollo

Aunque la configuración del entorno de trabajo se describe en el apéndice A, es necesario puntualizar que para la correcta implementación del controlador se requiere:

- Disponer de una máquina virtual con las herramientas especificadas en 3.2 y la versión actualizada de las mismas.

- Lanzar el controlador en un terminal independiente e instalar el esqueleto de la aplicación. En concreto, al partir de un switch inteligente que almacena las direcciones en el controlador, se activa dicho switch tal y como muestra la imagen 5.8. Posteriormente se comprueba que esté activo y se configura el método de depuración. Con el último comando mostrado, se puede aplicar un cierto filtro para depurar el programa más fácilmente a lo largo de la implementación.

- Iniciar el emulador y lanzar la topología sobre la que actuará el controlador. Como la implementación no requiere un escenario excesivamente complejo, se lanza la topología más básica que permite realizar experimentos de funcionamiento, disponiendo de: un controlador remoto, sobre el que se programa la aplicación; un switch, que conecta dicho controlador con los hosts; y tres hosts, tal y como muestra la figura 5.9. Para ello, se hace uso del comando:

  ```
  ubuntu@sdnhubvm:~$ sudo mn --topo single,3 --mac --switch ovsk,
  protocols=OpenFlow13 --controller remote
  ```

- Para el caso de IPv6, será necesario editar esa topología una vez creada. Siguiendo los pasos detallados en el apéndice A sobre la configuración de IPv6, se configura uno de los hosts del escenario para que funcione como router de la red, tal y como muestra la figura 5.10.

  Respecto a su configuración en Mininet, para la última versión del emulador, disponible desde Abril de 2015, las direcciones IPv6 de enlace están automáticamente configuradas de acuerdo con la dirección

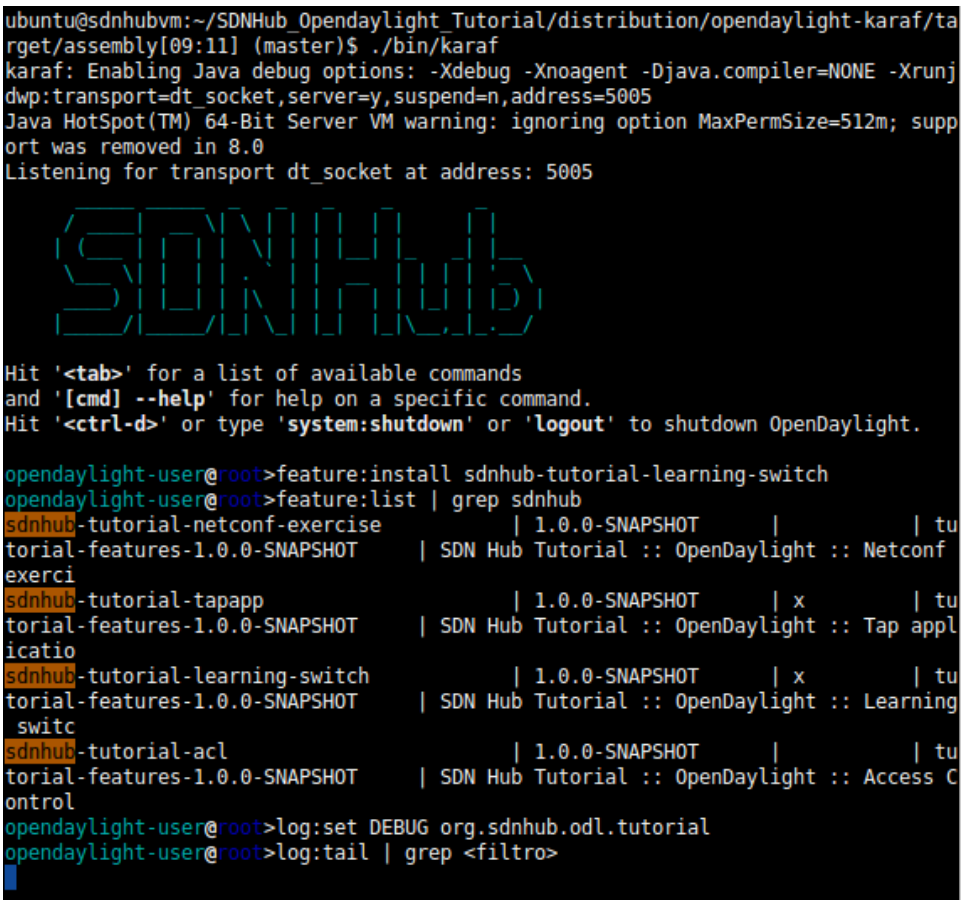


Figura 5.8: Puesta en marcha del controlador SDN e instalación de la aplicación.

MAC de cada host. Habrá que realizar sin embargo un proceso de configuración para las direcciones globales, tal y como se explica en el apéndice A.

- Lanzar Wireshark para analizar el tráfico de la red, así como Eclipse para editar el programa instalado en el controlador. Destacar por último que cada cambio realizado en dicho programa se actualizará una vez reinstalado Maven en el sistema y relanzado el controlador, por lo que estos pasos serán necesarios al inicio cada vez que se modifique el código.

5.3.2. Detección de paquetes

Tal y como se ha diseñado, la primera fase de la aplicación es la detección de paquetes. Se diferencian tres tipos de paquetes (IPv4, IPv6 y ARP), y es

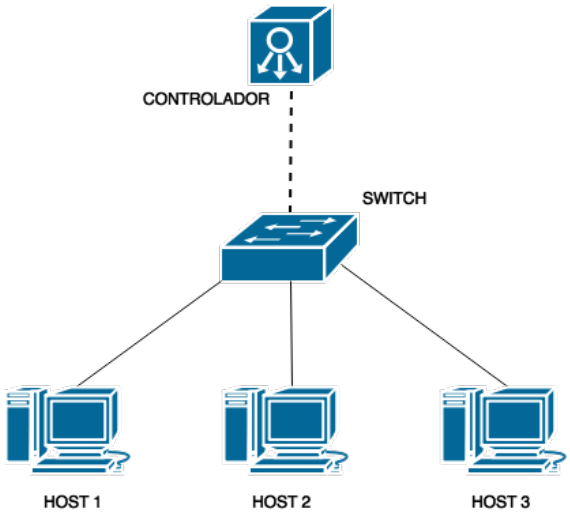


Figura 5.9: Escenario básico lanzado en Mininet para el proceso de implementación del filtro ARP para IPv4.

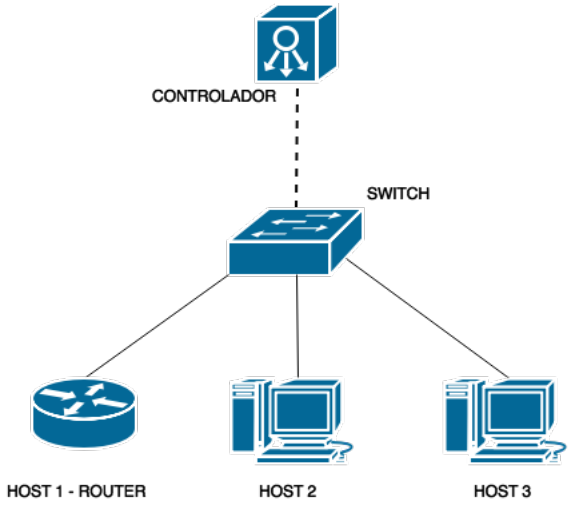


Figura 5.10: Escenario modificado para la implementación en IPv6.

necesario conocer su estructura para poder clasificarlos y realizar el filtrado posterior. Para diferenciar los paquetes existe la variable "tipo de Ethernet", que se extrae del *payload* automáticamente con el comando

```
byte[] etherTypeRaw = PacketParsingUtils.extractEtherType
(notification.getPayload());
```

Caso IPv4 - Tipo Ethernet 0x0800

Esta opción es la más fácil de implementar, ya que tan sólo almacena la información del nodo origen del mensaje en caso de que no exista una entrada en el controlador o que la información disponible esté desactualizada.

No se profundiza mucho por tanto en el resto de componentes del paquete, que tendrá una cabecera como la de la figura 5.11.

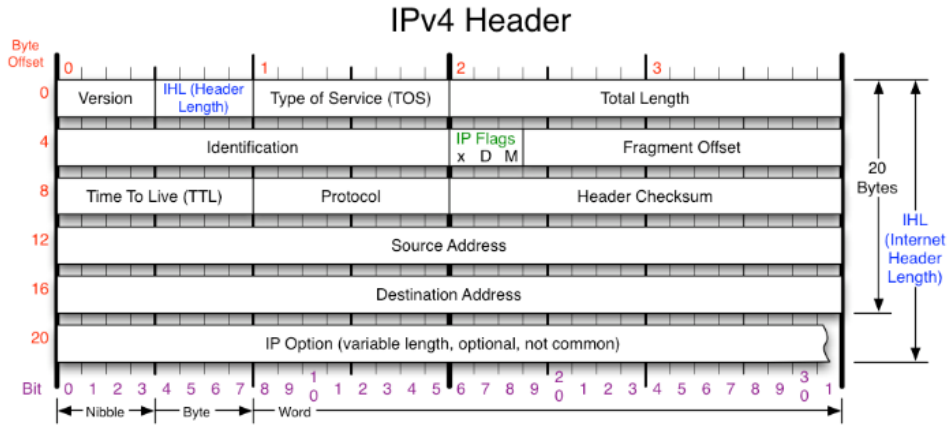


Figura 5.11: Cabecera de un mensaje IPv4. [35]

Así, tras extraer las direcciones física e IP del origen, se llama al método creado para la actualización de las entradas de la tabla, que a su vez llama al método para crear entradas, en caso de que éstas no hayan sido creadas previamente. Se presenta a continuación el código para ambos métodos. Mencionar que la clase "hostData"se ha creado para almacenar los datos de cada una de las entidades de la red.

```
public void updateNode(String sourceIP, String sourceMAC, byte[]
 sourcePayload, byte[] sourceMacB, byte[]sourceIpB){
     hostData nodeData = hostTable.get(sourceIP);
     if (nodeData == null){
     createNode(sourceIP, sourceMAC, sourcePayload, sourceMacB,
      sourceIpB);
     LOG.info("bvalera - Host has been added to the table.");
     } else if (System.currentTimeMillis() > (120000 +
     (nodeData.getDate()))){//2min
            createNode(sourceIP, sourceMAC, sourcePayload,
            sourceMacB, sourceIpB);
            LOG.info("bvalera - Host has been updated.");
}
    }
```

```
public void createNode(String sourceIP, String sourceMAC,
 byte[] sourcePayload, byte[]sourceMacB, byte[]sourceIpB){
     hostData nodeData = new hostData();
nodeData.setIp(sourceIP);
nodeData.setMac(sourceMAC);
nodeData.setDate();
nodeData.setPayload(sourcePayload);
nodeData.setMacB(sourceMacB);
nodeData.setIpB(sourceIpB);
hostTable.put(sourceIP, nodeData);
}
```

Caso ARP - Tipo Ethernet 0x0806

En el caso de ARP, se realiza también la fase de extracción de direcciones MAC e IP, aunque este caso se incluirá al destino, no solo al origen. Respecto a la creación de una entrada para el nodo origen, puede considerarse tanto parte de esta fase como de la de filtrado.

En cualquier caso, la cabecera para este caso es diferente de la presentada anteriormente, tal y como muestra la figura 5.12. Aquí, tras las direcciones físicas de destino y origen y el tipo de Ethernet, aparece el paquete ARP en sí. Destacar que este protocolo se utiliza para solicitar la dirección física correspondiente a una dirección IP dada, por lo que para la dirección física destino de las peticiones, en las que es desconocida, se utiliza broadcast, siendo esta dirección FF:FF:FF:FF:FF:FF.

Mencionar también que los bytes utilizados en el proceso de filtrado corresponden al código de operación, que permite distinguir las solicitudes de las respuestas.

Caso IPv6 - Tipo Ethernet 0x86DD

En este caso, además de extraer del *payload* las direcciones correspondientes al origen y al destino, se extrae el byte correspondiente a la siguiente cabecera, ya que tan sólo se filtran los mensajes ICMP, cuyo código es 0x3A. El estudio del paquete ICMP se realizará en la sección de filtrado, mientras que la figura 5.13 muestra la trama IPv6 propiamente dicha.

5.3.3. Filtrado de paquetes

De nuevo cabe diferenciar dos casos en el filtrado, de acuerdo con los resultados obtenidos en la sección anterior. Esta fase se encarga de clasificar

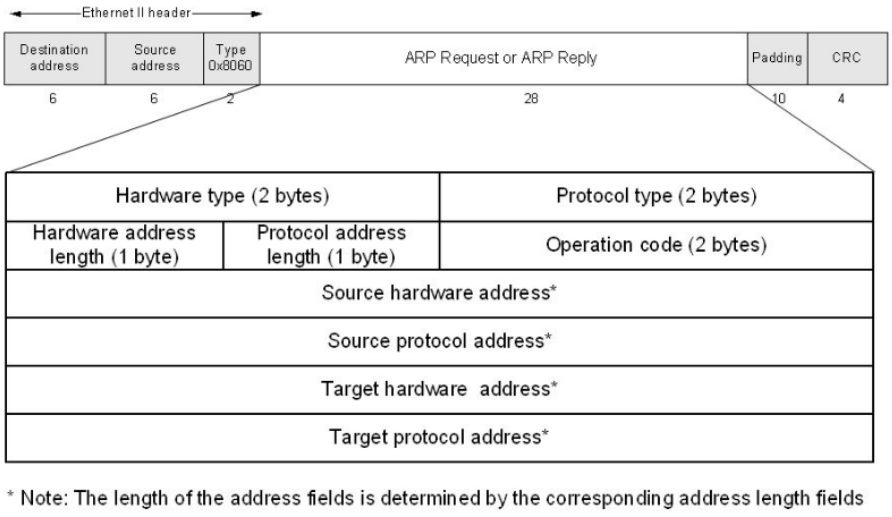


Figura 5.12: Formato de una trama ARP. [8]

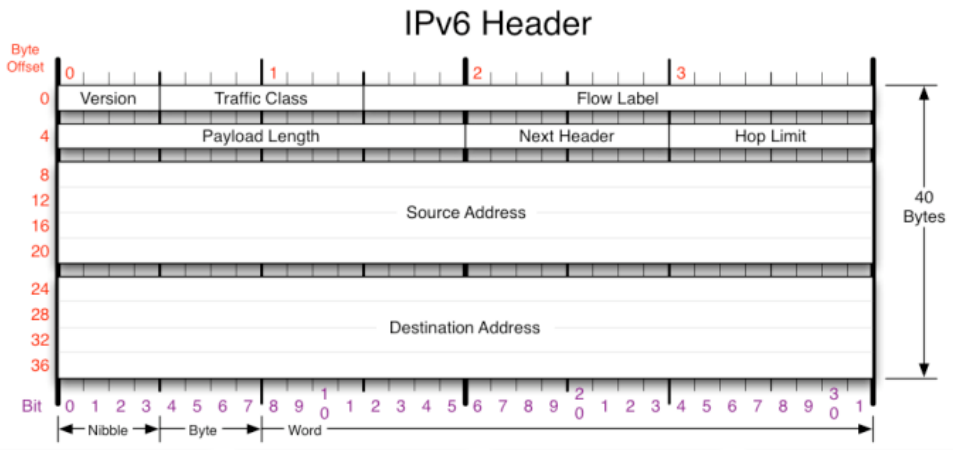


Figura 5.13: Cabecera de un mensaje IPv6. [12]

y analizar los mensajes de cada uno de los protocolos, extrayendo los datos necesarios para crear las entradas de la tabla, en caso de que no se haya hecho con anterioridad, y comprobando si se dispone de información necesaria para atender las solicitudes recibidas sin necesidad de permitir el reenvío de dichas solicitudes al resto de la red.

Caso ARP

Tal y como mostraba la imagen 5.12, los paquetes de este protocolo constan de 28 bytes dedicados únicamente a lo que es la solicitud/respuesta en sí. De esos 28 bytes, se considera que el tipo de Hardware será siempre el mismo, ya que opera dentro de la misma red, el tipo de protocolo será el correspondiente a IPv4, y la longitud de las direcciones MAC e IP serán también constantes. Respecto al código de operación, será 01 si se trata de una solicitud (*request*) y 02 si es una respuesta (*reply*). [41]

La idea es que el controlador sea capaz de generar respuestas automáticamente cuando disponga de la información necesaria sobre el nodo destino, de forma que los mensajes broadcast que se mandarían a toda la red se bloquean en el controlador, que contesta al origen directamente sin implicar al resto.

Para ello, se construye un "payload"de 42 bytes respuesta que consta de una parte variable y otra constante, siendo la variable la parte compuesta por las direcciones IP y física de origen y destino, teniendo en cuenta que el nodo origen de la respuesta debe ser el destino de la solicitud, tal y como muestran las tablas 5.3.3 y 5.3.3.

MAC destino	MAC origen	Parte constante request	
FF:FF:FF:FF:FF:FF	6 bytes	08 06 00 01 08 00 06 04 00 01	
MAC origen	IP origen	MAC destino	IP destino
6 bytes	4 bytes	00:00:00:00:00:00	4 bytes

Tabla 5.2: Payload de un paquete request en ARP.

MAC origen	MAC destino	Parte constante reply	
6 bytes	6 bytes	08 06 00 01 08 00 06 04 00 02	
MAC destino	IP destino	MAC origen	IP origen
6 bytes	4 bytes	6 bytes	4 bytes

Tabla 5.3: Payload generado en el controlador como reply ARP.

Caso IPv6

El caso de IPv6 es más complejo, ya que ICMP presenta cuatro posibilidades a filtrar. Así, tras los 40 primeros bytes del paquete, que componen la cabecera de IPv6 de donde se extraen las direcciones de origen y destino, se evaluará el byte correspondiente al tipo, mostrado en la imagen 5.14, que diferencia entre anuncio de router, solicitud de router, anuncio de vecino y solicitud de vecino.

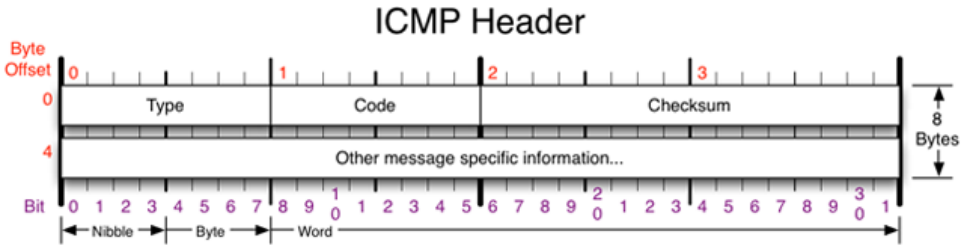


Figura 5.14: Formato de los mensajes ICMP. [37]

Así, con el primer mensaje de anuncio de router (tipo 0x86), se almacenará el payload del mismo, que contiene el prefijo de red para la asignación de direcciones. En este caso, se vuelven a utilizar los métodos presentados anteriormente de creación de entradas, aunque en este caso el tiempo considerado para la actualización de la entrada es mayor que en el caso de ARP, ya que la probabilidad de que un nodo cambie de red es mayor que la de que lo haga un router.

De esta forma, cuando se reciba una solicitud de router (tipo 0x85), se comprobará si existe el *payload* asociado al router de esa red y, de ser así, se responde automáticamente al nodo solicitante, sin necesidad de continuar el reenvío de la solicitud al resto de la red. Para ello, se utiliza el siguiente código:

```
if (payloadRouterB != null){
            packetOut(ingressNodeRef, ingressNodeConnectorRef,
            payloadRouterB);
                        LOG.info("bvalera - Controller sends
                        RA itself.");
                        boolOut = false;
            }
```

La importancia de la variable "boolOut"se presenta en el apartado 5.3.4. Por otra parte, el método de envío de paquetes se presenta a continuación:

```
private void packetOut(NodeRef egressNodeRef, NodeConnectorRef
egressNodeConnectorRef, byte[] payload) {
        Preconditions.checkNotNull(packetProcessingService);
        LOG.debug("Flooding packet of size {} out of port {}",
        payload.length, egressNodeConnectorRef);

        //Construct input for RPC call to packet processing service
        TransmitPacketInput input = new TransmitPacketInputBuilder()
                .setPayload(payload)
```

```
                .setNode(egressNodeRef)
                .setEgress(egressNodeConnectorRef)
                .build();
        packetProcessingService.transmitPacket(input);
    }
```

De forma similar, cuando se realiza una solicitud de vecino (tipo 0x87) y tras actualizar la entrada del nodo solicitante, se busca la entrada correspondiente al nodo destino en la tabla de la siguiente forma:

```
hostData neighborDest = hostTable.get(dstAddress);
         if (neighborDest != null){
         packetOut(ingressNodeRef, ingressNodeConnectorRef,
         neighborDest.getNodePayload());
                             LOG.info("bvalera - Controller sends
                             NA itself.");
                             boolOut = false;
         }
```

Así, de disponer de la información necesaria para responder por sí mismo, el controlador se encarga de bloquear el mensaje de solicitud y atender la petición sin necesidad de propagar dicho mensaje al resto de la red.

El anuncio de vecino (tipo 0x88) será simplemente una creación o actualización de la entrada correspondiente al nodo origen cuando sea necesario, o una sentencia "boolOut = false.[en] caso de que la entrada esté actualizada.

5.3.4. Reenvío de paquetes

El reenvío de paquetes es la fase más sencilla de implementar, ya que, tal y como muestra el código a continuación, tiene en cuenta una variable booleana definida durante el filtrado.

De ser "true", se realiza un reenvío, bien al destino en caso de existir, o a toda la red de no haberse determinado un destino concreto. En caso de que el controlador haya filtrado el mensaje, esta variable se establece "false", por lo que no habrá reenvío del paquete. En su lugar, se habrá realizado la elaboración y envío de una respuesta en el propio método de filtrado, de forma que el controlador se encarga de gestionar la petición sin implicar al resto de la red.

```
if (boolOut){
             LOG.info("bvalera - packetOut with boolOut true");
             if (egressNodeConnectorId != null) {
```

```
            programL2Flow(ingressNodeId, dstMac,
            ingressNodeConnectorId, egressNodeConnectorId);
            NodeConnectorRef egressNodeConnectorRef =
            InventoryUtils.getNodeConnectorRef
            (egressNodeConnectorId);
            packetOut(ingressNodeRef, egressNodeConnectorRef,
            payload);
        } else {
         //2.3.2 Flood packet
            packetOut(ingressNodeRef, floodNodeConnectorRef,
            payload);
        }
    }
```

Capítulo 6

Evaluación de la solución

En este capítulo se presenta la fase de pruebas y resultados obtenidos para evaluar la solución implementada. Se distinguen cuatro pruebas, con un objetivo específico y un entorno experimental propio cada una.

En primer lugar, se comprobará el correcto funcionamiento del controlador implementado para IPv4 e IPv6. Posteriormente, se comprueba el funcionamiento del sistema cuando se producen mensajes en ambos protolos simultáneamente. Por último, se estudiará el rendimiento del sistema y la mejora que supone el controlador cuando se producen inundaciones en la red.

6.1. Funcionamiento con IPv4

6.1.1. Entorno experimental

Para comprobar el correcto funcionamiento del controlador en este caso basta con un escenario como el utilizado en el proceso de implementación, con ODL como controlador remoto, un switch OpenFlow y tres hosts, tal y como muestra la figura 6.1. Para ello, tras lanzar el controlador e instalar la aplicación, se hace uso del comando:

```
ubuntu@sdnhubvm:~$ sudo mn --topo single,3 --mac --switch ovsk,
protocols=OpenFlow13 --controller remote
```

Y se añade una regla para que el flujo del switch se dirija al controlador, con el comando:

```
mininet> s1 ovs-ofctl add-flow tcp:127.0.0.1:6634 -OOpenFlow13
priority=1,action=output:controller
```

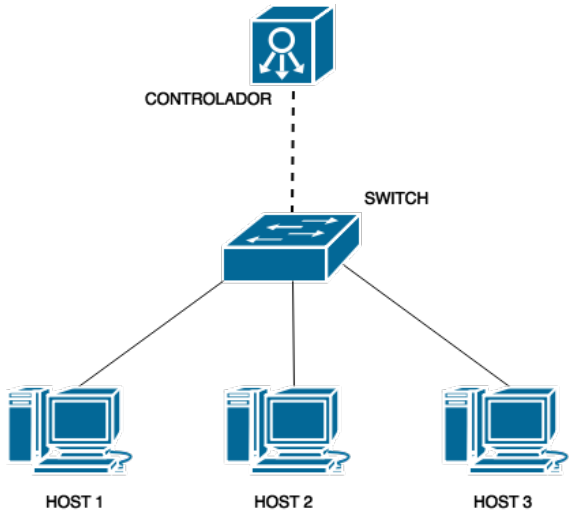


Figura 6.1: Escenario para comprobar el funcionamiento con IPv4 y switch OpenFlow.

6.1.2. Descripción del experimento

Esta prueba sirve para analizar el funcionamiento del controlador cuando se produce broadcast para ARP. Así, tras iniciar la red, disponiendo de 3 hosts sin entradas en la tabla del controlador, se realiza un "ping"del host h1 al host h2. Básicamente, al realizar un "ping"se envía un mensaje ICMP desde el nodo origen de tipo *echo request*, que el nodo destino deberá responder con un *echo reply*. En ese momento, se produce una petición ARP *request* con origen en h1 y destino en h2. Como respuesta, h2 genera un mensaje ARP *reply* con destino h1. En 6.2 se muestra este proceso. El intervalo señalado como 1 es en el que el controlador almacena las direcciones del nodo origen y lo registra en la tabla, mientras que en el intervalo 2 lo hace con el nodo destino, añadiendo la parte permanente del payload al registro para la construcción posterior de respuestas automáticas.

De esta forma, ambos hosts habrán quedado reconocidos y registrados en el controlador junto con la información relevante para que éste sea capaz de generar respuestas automáticas para atender futuras peticiones.

De igual forma, se realiza otro ping de h1 hacia h3. Llegados a este punto, si h2 realizase un ping con destino h3, habría tres posibilidades:

a. De no funcionar correctamente el filtrado, no sería posible completar la petición, ya que el controlador podría reconocer las entradas pero no generar una respuesta apropiada.

b. De no realizar el filtrado, h3 respondería con un ARP *reply* a h2. El

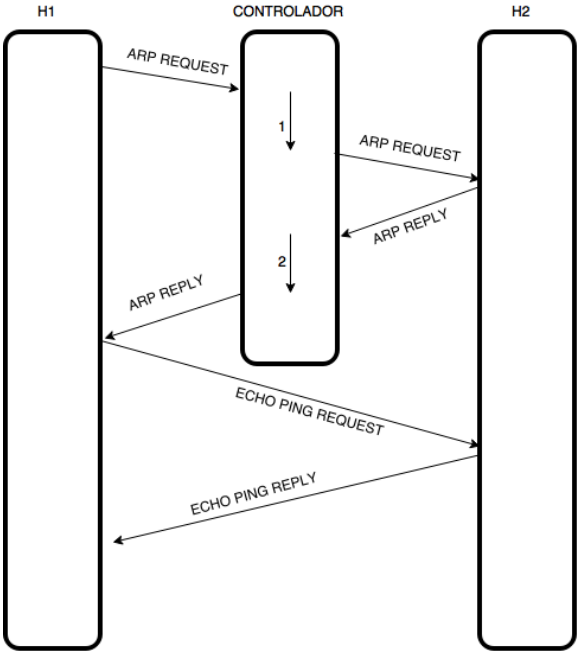


Figura 6.2: Diagrama de secuencias de la realización del "ping".

ping original *request* se habría enviado como broadcast al resto de la red, generando dicha respuesta por parte de h3.

c. De haber filtrado correctamente los mensajes, el controlador bloquea el broadcast de h2 y responde automáticamente con la dirección física asociada a h3. Por tanto, la comunicación entre nodos sería posible sin necesidad tampoco un mensaje ARP *reply* desde h3.

Para la realización de "ping.[en] Mininet para IPv4, basta con:

```
mininet> <nodo origen> ping <nodo destino>
```

Por ejemplo, en este caso:

```
mininet> h1 ping h2
mininet> h1 ping h3
mininet> h2 ping h3
```

6.1.3. Resultados obtenidos

Como era de esperar, no sólo es posible hacer el ping desde h2 hacia h3, lo cual se muestra en la captura 6.3, sino que el controlador es el encargado de

generar la respuesta. Esto se observa en la captura del terminal del controlador 6.4, en la que se muestra cómo la petición es contestada directamente por el controlador, de forma que se crea un *payload* respuesta y el siguiente paquete no sería un *reply*, como ocurre cuando no hay intervención del controlador. También se puede ver en las capturas realizadas del analizador de tráfico Wireshark: 6.5, que muestra la intervención del controlador a modo de respuesta frente a la petición, y 6.6, que desglosa el mensaje señalado en la captura anterior, mostrando cómo lo que envía el controlador es un mensaje ARP *reply*.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1644 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=811 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=376 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.492 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.177 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.104 ms
^C64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.334 ms

--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6006ms
rtt min/avg/max/mdev = 0.104/404.778/1644.663/580.377 ms, pipe 2
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=435 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.512 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.097 ms
^C
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 0.097/145.264/435.183/205.003 ms
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=583 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.38 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.174 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 0.174/146.224/583.154/252.262 ms
mininet>
```

Figura 6.3: Terminal de Mininet realizando ping.

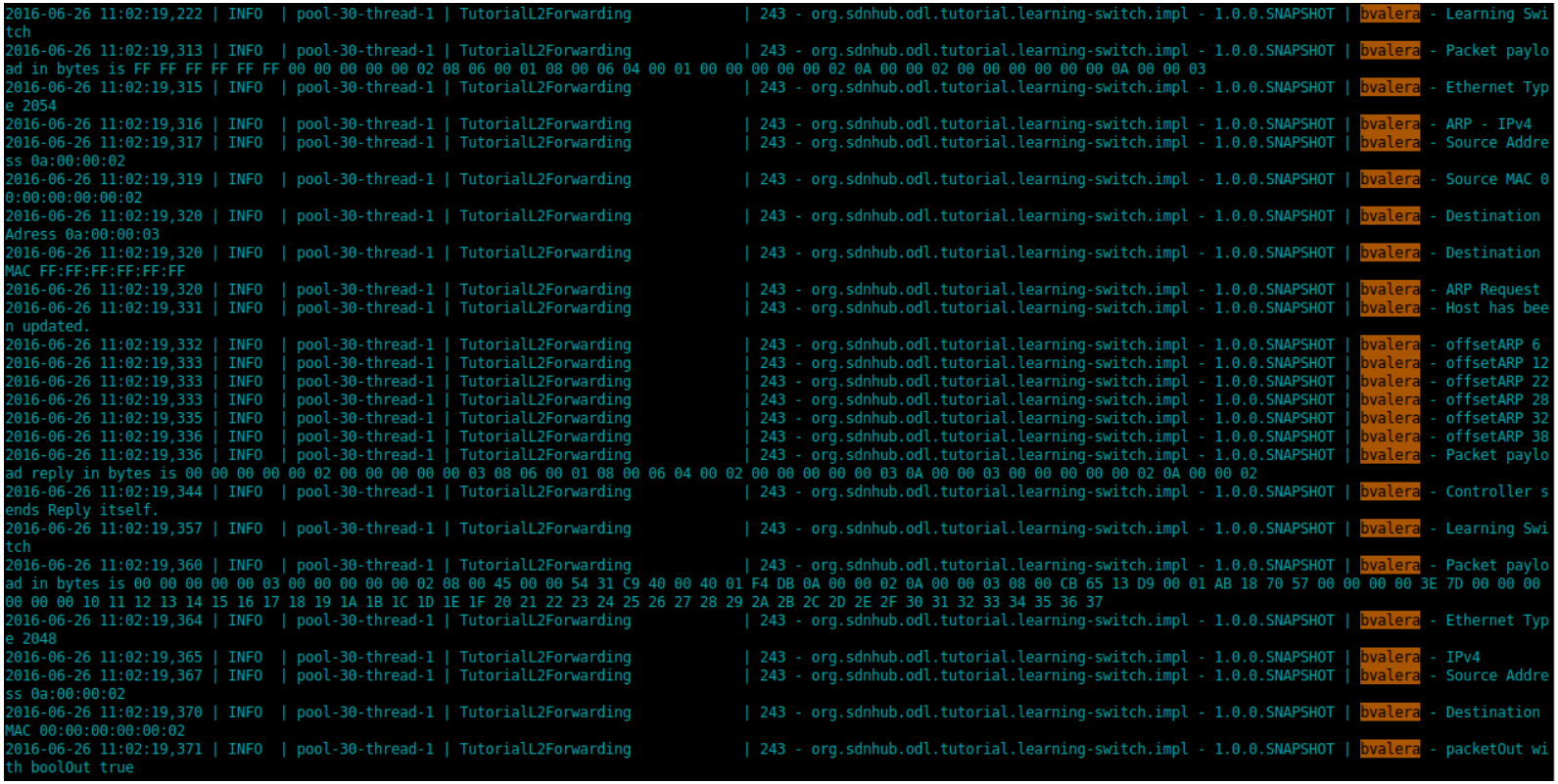


Figura 6.4: Terminal del controlador Karaf analizando IPv4 ARP Request.

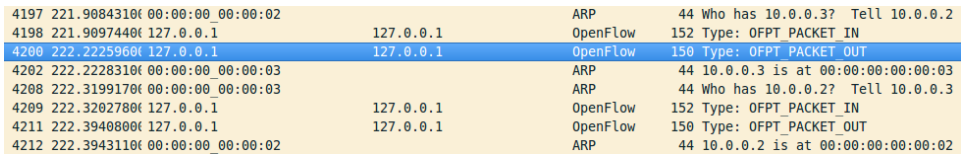


Figura 6.5: Captura Wireshark con la intervención del controlador bloqueando los paquetes broadcast.

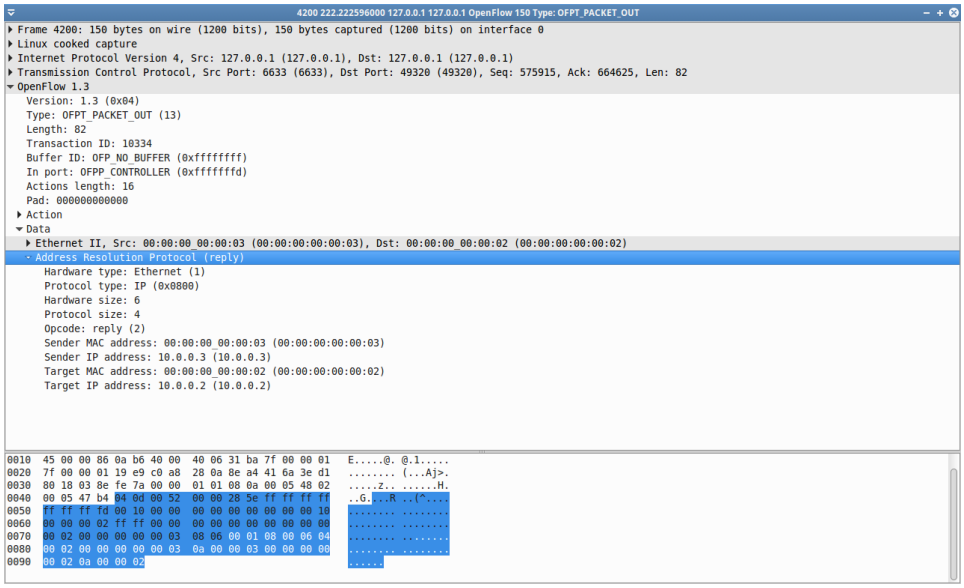


Figura 6.6: Mensaje respuesta generado por el controlador.

6.2. Funcionamiento con IPv6

6.2.1. Entorno experimental

De igual forma, será necesario realizar la comprobación del funcionamiento para el caso de IPv6, de nuevo con la topología utilizada en la implementación, aunque las pruebas en este caso difieren del caso anterior. Esto se debe al nuevo escenario, en el que se dispone de dos hosts y un router, además del switch OpenFlow y del controlador, tal y como muestra la figura 6.7.

Mencionar también que en este caso no es sólo la implementación del controlador en sí lo que se está corroborando, sino también la configuración de IPv6, ya que en este escenario un host sería el encargado de funcionar como router, asignando al resto de elementos de la red sus direcciones globales con el prefijo 2001:db8::/64. Es necesario por tanto realizar la configuración

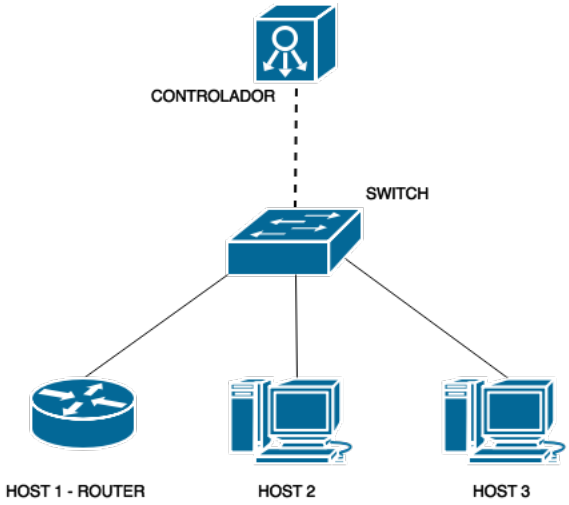


Figura 6.7: Escenario para comprobar el funcionamiento con IPv6 y switch OpenFlow.

en los hosts de Mininet una vez lanzado, tal y como se explica en el apéndice A.

6.2.2. Descripción del experimento

Este experimento, como ya se ha mencionado, comprueba el correcto funcionamiento del filtrado para la autoconfiguración del protocolo IPv6 en el sistema. Para ello se comprueba que, una vez realizadas la configuraciones, es posible realizar ping entre los dos hosts de la red, h2 y h3.

Si se elimina una de las interfaces, como la que enlaza h3 con el resto de la red, no sería posible como es lógico repetir ese ping, aunque si se vuelve a habilitar esa interfaz se plantea una situación con tres posibilidades:

a. De no realizar el filtrado correctamente, al añadirse un "nuevo"host a la red sería necesario realizar la solicitud de router para autoasignarse una dirección, junto con el anuncio correspondiente que contiene el prefijo de la red, así como una solicitud de vecino para evitar la duplicidad de direcciones y un anuncio de vecino para notificar al resto de la red que ha sido añadido.

b. De no funcionar correctamente, no sería posible completar la petición, ya que el controlador podría reconocer la entrada pero no generar una respuesta apropiada al host h3, por lo que éste no se asignaría la IP correctamente y no podría ponerse en contacto con el resto de nodos vecinos.

c. De haber filtrado correctamente los mensajes, el controlador bloquea el tráfico innecesario y responde automáticamente a h3, por lo que no sólo filtra esos mensajes de solicitud y anuncio, sino que proporciona la información necesaria para que se vuelva a producir comunicación entre h3 y el resto de la red.

Los pasos en resumen serían:

```
mininet> h2 ping6 2001:db8::200:ff:fe00:3 -I h2-eth0
mininet> link s1 h3 down
mininet> link s1 h3 up
mininet> h2 ping6 2001:db8::200:ff:fe00:3 -I h2-eth0
```

6.2.3. Resultados obtenidos

De nuevo, tal y como se preveía, inicialmente se puede ejecutar la herramienta ping6, tras deshabilitar la interfaz no es posible y, tras volverla a habilitar, vuelve a ser posible ejecutar ping6 entre los hosts implicados. En la captura 6.8, se observa cómo se realiza este proceso, junto con el resultado esperado. Por otra parte, la captura 6.9 muestra el terminal del host h3. Se puede observar cómo al deshabilitar la interfaz este host pierde su dirección global IPv6, que posteriormente recupera automáticamente al volver a habilitarse la interfaz.

Además, en el terminal del controlador 6.10 se observa cómo éste mismo responde la solicitud de router de h3, de forma que continúa la comunicación entre nodos sin necesidad de que el sistema atienda esa solicitud.

```
mininet> h2 ping6 2001:db8::200:ff:fe00:3 -I h2-eth0
PING 2001:db8::200:ff:fe00:3(2001:db8::200:ff:fe00:3) from 2001:db8::200:ff:fe00:2 h2-eth0: 56 data bytes
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=1 ttl=64 time=8.58 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=2 ttl=64 time=0.082 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=3 ttl=64 time=0.140 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=4 ttl=64 time=0.086 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=5 ttl=64 time=0.090 ms
^C
--- 2001:db8::200:ff:fe00:3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.082/1.796/8.583/3.393 ms
mininet> link s1 h3 down
mininet> h2 ping6 2001:db8::200:ff:fe00:3 -I h2-eth0
PING 2001:db8::200:ff:fe00:3(2001:db8::200:ff:fe00:3) from 2001:db8::200:ff:fe00:2 h2-eth0: 56 data bytes
From 2001:db8::200:ff:fe00:2 icmp_seq=9 Destination unreachable: Address unreachable
From 2001:db8::200:ff:fe00:2 icmp_seq=10 Destination unreachable: Address unreachable
From 2001:db8::200:ff:fe00:2 icmp_seq=11 Destination unreachable: Address unreachable
^C
--- 2001:db8::200:ff:fe00:3 ping statistics ---
13 packets transmitted, 0 received, +3 errors, 100% packet loss, time 12088ms

mininet> link s1 h3 up
mininet> h2 ping6 2001:db8::200:ff:fe00:3 -I h2-eth0
PING 2001:db8::200:ff:fe00:3(2001:db8::200:ff:fe00:3) from 2001:db8::200:ff:fe00:2 h2-eth0: 56 data bytes
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=1 ttl=64 time=3.60 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=2 ttl=64 time=0.123 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=3 ttl=64 time=0.135 ms
64 bytes from 2001:db8::200:ff:fe00:3: icmp_seq=4 ttl=64 time=0.110 ms
^C
--- 2001:db8::200:ff:fe00:3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.110/0.992/3.600/1.505 ms
mininet>
```

Figura 6.8: Terminal de Mininet realizando el proceso de ping6, deshabilitar la interfaz de h3, habilitarla y ping6 de nuevo, respectivamente.

```
root@sdnhubvm:~[03:18]$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[03:23]$ ifconfig
h3-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:03
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: 2001:db8::200:ff:fe00:3/64 Scope:Global
          inet6 addr: fe80::200:ff:fe00:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:245 errors:0 dropped:0 overruns:0 frame:0
          TX packets:37 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:21339 (21.3 KB)  TX bytes:3242 (3.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[03:24]$
```

Figura 6.9: Terminal de h3 cuando la interfaz está deshabilitada (sin direcciones IPv6) y posteriormente habilitada.

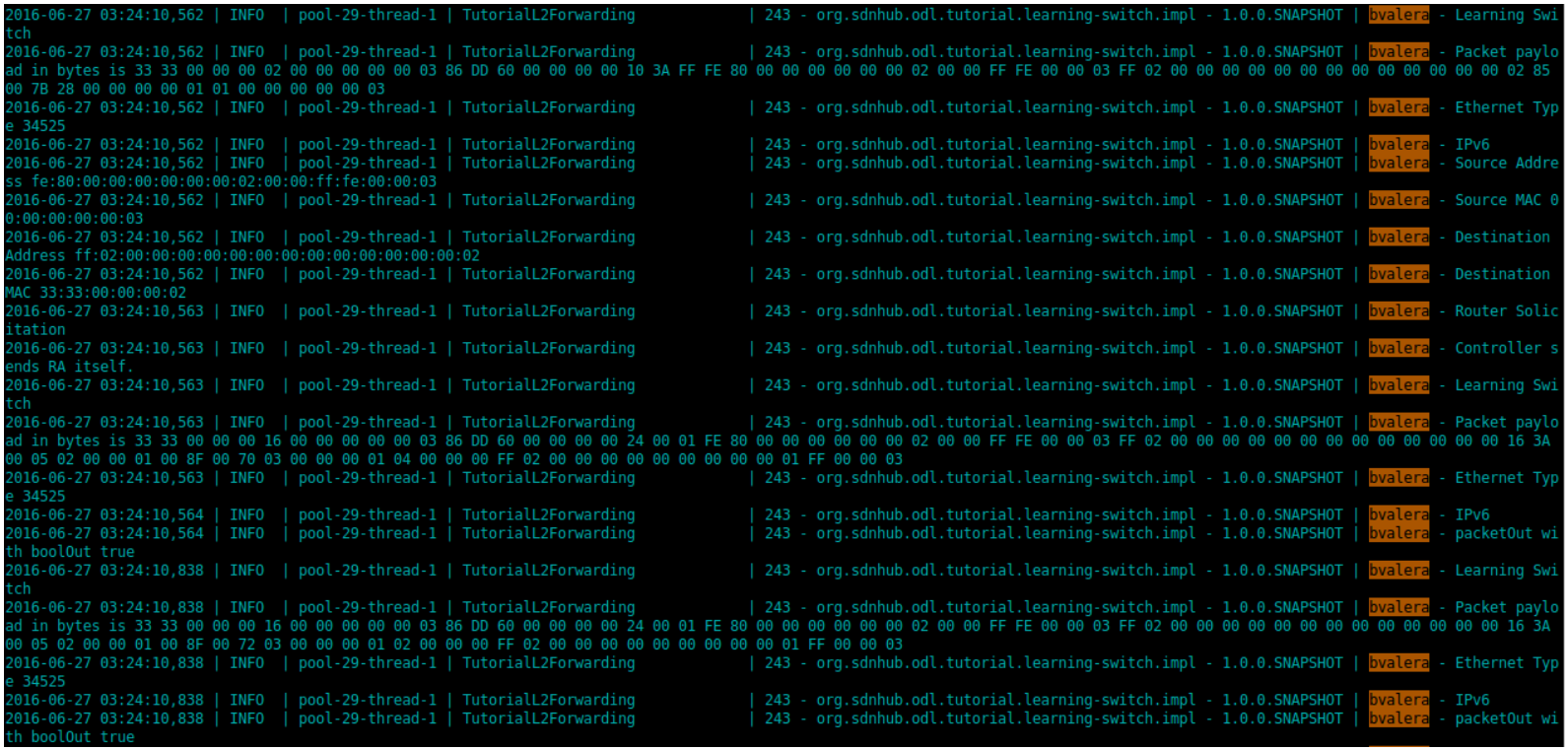
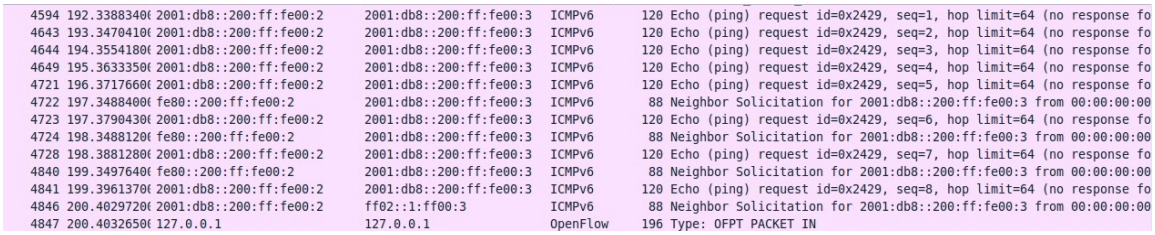


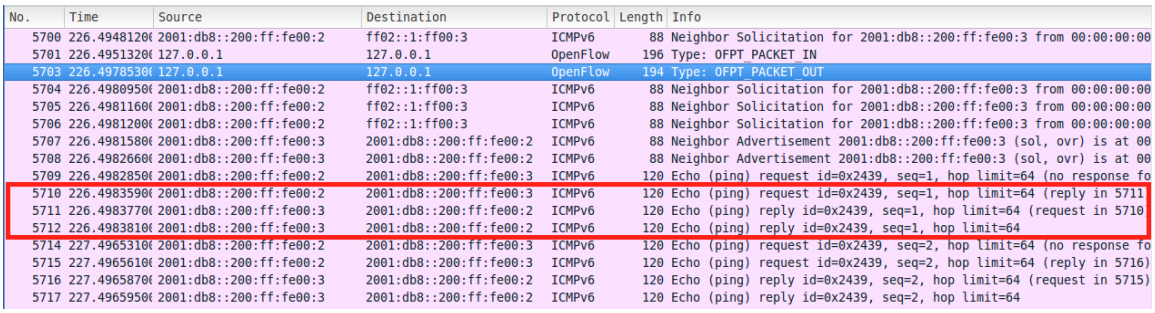
Figura 6.10: Terminal del controlador Karaf respondiendo automáticamente la solicitud de router de h3.

En Wireshark no es tan sencillo verlo, pero se puede apreciar cómo con la interfaz deshabilitada el ping6 no es posible en la captura 6.11. Posteriormente interviene el controlador generando un mensaje desde el host h2, que se pone en contacto con h3 para posibilitar el ping6, tal y como muestra la captura 6.12. Este mensaje generado aparece en 6.13, mostrando cómo aunque es enviado por el controlador el nodo origen es h2, lo que explica por qué aparece en la captura anterior que el nodo h2 está enviando el mensaje, ya que Wireshark lo interpreta como dos mensajes independientes.



4594	192.3388340(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=1, hop limit=64 (no response fo
4643	193.3470410(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=2, hop limit=64 (no response fo
4644	194.3554180(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=3, hop limit=64 (no response fo
4649	195.3633350(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=4, hop limit=64 (no response fo
4721	196.3717660(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=5, hop limit=64 (no response fo
4722	197.3488400(	fe80::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
4723	197.3790430(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=6, hop limit=64 (no response fo
4724	198.3488120(	fe80::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
4728	198.3881280(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=7, hop limit=64 (no response fo
4840	199.3497640(	fe80::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
4841	199.3961370(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2429, seq=8, hop limit=64 (no response fo
4846	200.4029720(	2001:db8::200:ff:fe00:2	ff02::1:ff00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
4847	200.4032650(	127.0.0.1	127.0.0.1	OpenFlow	196	Type: OFPT_PACKET_IN

Figura 6.11: Captura Wireshark con la interfaz deshabilitada. No se pueden atender las peticiones echo ping request.



No.	Time	Source	Destination	Protocol	Length	Info
5700	226.4948120(	2001:db8::200:ff:fe00:2	ff02::1:ff00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
5701	226.4951320(	127.0.0.1	127.0.0.1	OpenFlow	196	Type: OFPT_PACKET_IN
5703	226.4978530(	127.0.0.1	127.0.0.1	OpenFlow	194	Type: OFPT_PACKET_OUT
5704	226.4980950(	2001:db8::200:ff:fe00:2	ff02::1:ff00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
5705	226.4981160(	2001:db8::200:ff:fe00:2	ff02::1:ff00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
5706	226.4981200(	2001:db8::200:ff:fe00:2	ff02::1:ff00:3	ICMPv6	88	Neighbor Solicitation for 2001:db8::200:ff:fe00:3 from 00:00:00:00
5707	226.4981580(	2001:db8::200:ff:fe00:3	2001:db8::200:ff:fe00:2	ICMPv6	88	Neighbor Advertisement 2001:db8::200:ff:fe00:3 (sol, ovr) is at 00
5708	226.4982660(	2001:db8::200:ff:fe00:3	2001:db8::200:ff:fe00:2	ICMPv6	88	Neighbor Advertisement 2001:db8::200:ff:fe00:3 (sol, ovr) is at 00
5709	226.4982850(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2439, seq=1, hop limit=64 (no response fo
5710	226.4983590(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2439, seq=1, hop limit=64 (reply in 5711
5711	226.4983770(	2001:db8::200:ff:fe00:3	2001:db8::200:ff:fe00:2	ICMPv6	120	Echo (ping) reply id=0x2439, seq=1, hop limit=64 (request in 5710
5712	226.4983810(	2001:db8::200:ff:fe00:3	2001:db8::200:ff:fe00:2	ICMPv6	120	Echo (ping) reply id=0x2439, seq=1, hop limit=64
5714	227.4965310(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2439, seq=2, hop limit=64 (no response fo
5715	227.4965610(	2001:db8::200:ff:fe00:2	2001:db8::200:ff:fe00:3	ICMPv6	120	Echo (ping) request id=0x2439, seq=2, hop limit=64 (reply in 5716)
5716	227.4965870(	2001:db8::200:ff:fe00:3	2001:db8::200:ff:fe00:2	ICMPv6	120	Echo (ping) reply id=0x2439, seq=2, hop limit=64 (request in 5715)
5717	227.4965950(	2001:db8::200:ff:fe00:3	2001:db8::200:ff:fe00:2	ICMPv6	120	Echo (ping) reply id=0x2439, seq=2, hop limit=64

Figura 6.12: Captura Wireshark con la intervención del controlador. Ahora sí es posible ejecutar ping.

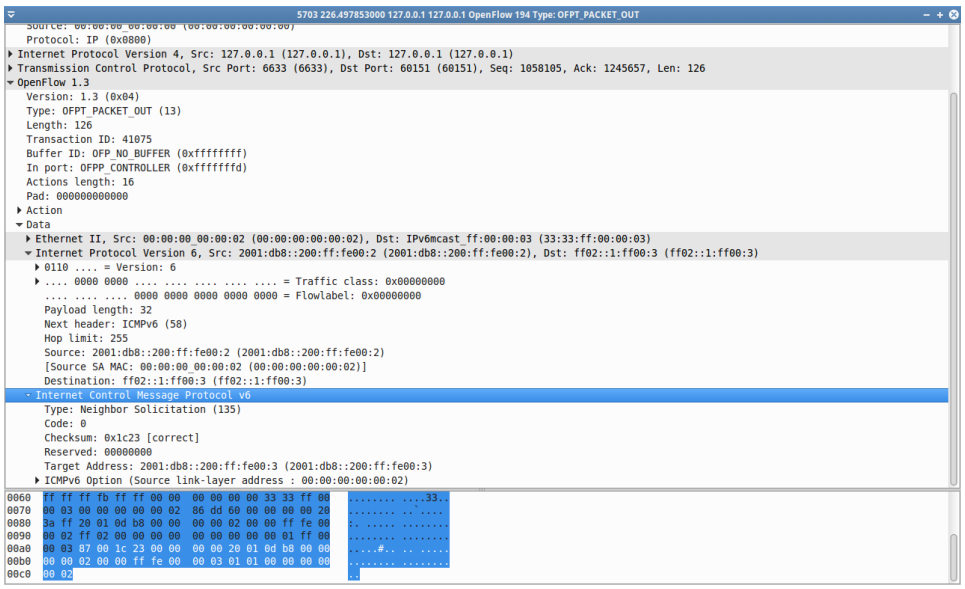


Figura 6.13: Mensaje respuesta generado por el controlador para ICMPv6.

6.3. Funcionamiento conjunto de IPv4 e IPv6

6.3.1. Entorno experimental

En este caso, ya que se estudia el funcionamiento conjunto, se volverá a utilizar la topología presentada en la figura 6.7. Aunque h1 funciona como router para IPv6, sí que puede seguir considerándose un host en IPv4, por lo que se puede realizar esta prueba sin complicar más el escenario de estudio.

6.3.2. Descripción del experimento

Se comprueba en este caso que el filtro del controlador puede trabajar simultáneamente para ARP y para IPv6 indistintamente. Para ello, se vuelve a realizar ping de h2 a h1 y a h3 y se deshabilita el enlace de h3 con el switch. Posteriormente, se vuelve a habilitar y, mientras el protocolo ICMP está enviando sus mensajes de arranque, se enviará un ping de h1 a h3 y un ping6 de h3 a h2 mediante un script creado previamente con los comandos:

```
mininet> h1 ping h3 &
mininet> h3 ping6 2001:db8::200:ff:fe00:2 -I h3-eth0 &
```

Dicho script se puede cargar desde Mininet una vez lanzada la topología con el comando:

```
mininet> source script
```

6.3.3. Resultados obtenidos

De nuevo los resultados son acordes a las expectativas y es posible utilizar conjuntamente IPv4 e IPv6. Esto permitirá también evaluar el rendimiento del sistema al completo, lo cual se presenta en la siguiente sección.

En concreto, se puede observar en el terminal del controlador mostrado en 6.15 cómo realiza el filtrado de un anuncio de router de un router añadido a la red con anterioridad y seguidamente responde automáticamente una petición ARP. Además, Wireshark muestra ambos protocolos realizando ping y ping6 al mismo tiempo, junto con las intervenciones del controlador, tal y como refleja la captura 6.14.

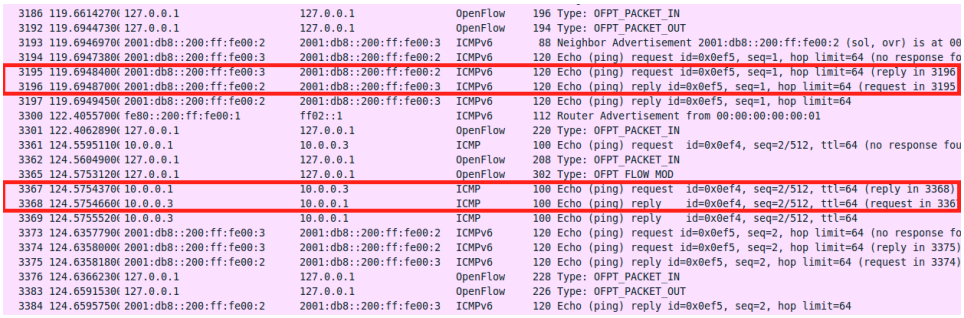


Figura 6.14: Captura Wireshark con filtro para ambos protocolos. Se pueden observar echo ping request y reply para IPv4 e IPv6.

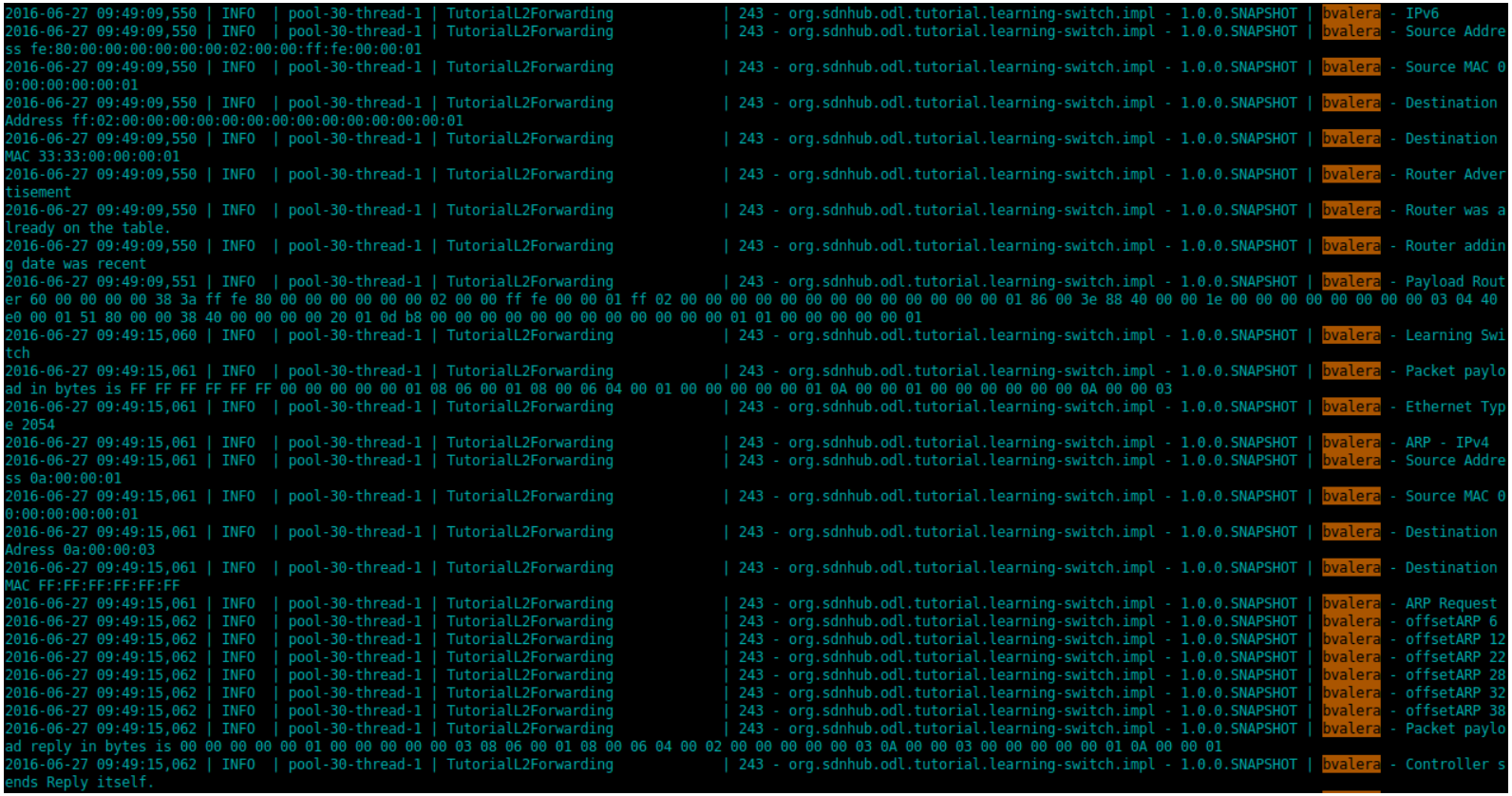


Figura 6.15: Terminal del controlador respondiendo solicitudes de ambos protocolos.

6.4. Rendimiento del sistema

Por último, se realizará un estudio de la mejora real que supone el desarrollo de esta solución. Previamente se ha analizado que funcionara en cada uno de los escenarios planteados y de acuerdo a lo esperado, aunque el objetivo inicial era mejorar el rendimiento de este sistema filtrando el tráfico de la red, lo que se analiza con esta última prueba.

6.4.1. Entorno experimental

Se utilizará la topología básica utilizada para la implementación de IPv6 para realizar este experimento, mostrada en 5.10, variando sin embargo el número de nodos. Es necesario por tanto realizar la configuración de este protocolo tal y como se indica en el apéndice A en ambas máquinas, la que contiene la aplicación del controlador y la que utiliza el controlador a modo de *Learning Switch* normal. Este switch viene implementado en la imagen por defecto y es parte del esqueleto inicial de la aplicación implementada, ya que es capaz de "aprender"las direcciones MAC asociadas a cada puerto por el que alcanzarlas, simulando el comportamiento de un switch Ethernet, aunque no realiza ningún tipo de filtrado.

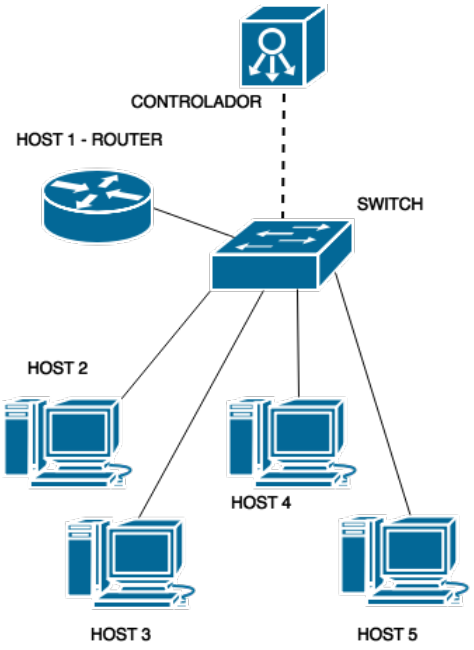


Figura 6.16: Escenario inicial con 5 nodos y un switch OpenFlow.

En concreto, se prueban los escenarios 6.16, 6.17 y 6.18, con 5, 10 y 20 nodos respectivamente. Sin embargo, para el caso del sistema con 20 nodos

y el controlador actuando como *Learning Switch*, no fue posible completar la prueba, ya que el sistema se bloqueaba y reiniciaba la máquina por completo, llegando a impedir el funcionamiento del equipo anfitrión.

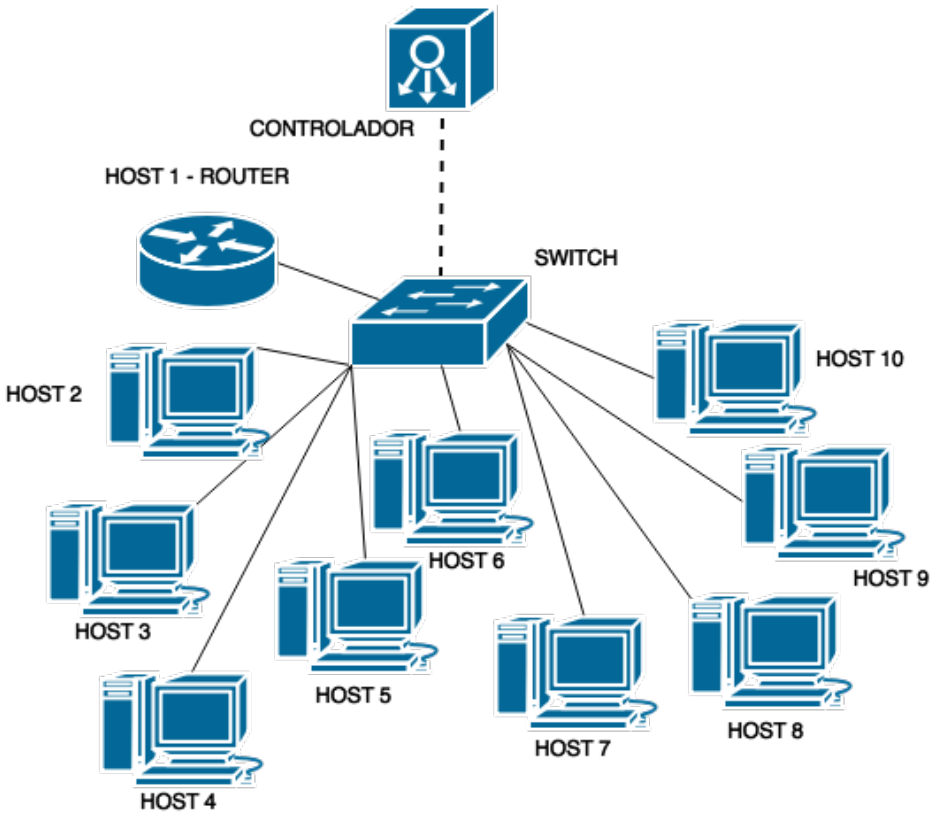


Figura 6.17: Escenario final con 10 nodos y un switch OpenFlow.

6.4.2. Descripción del experimento

En esta última prueba, se estudia el número de paquetes transmitido perteneciente a los protocolos ARP e ICMPv6 en una red que transmite 20 paquetes de cada protocolo por nodo con un intervalo de 5 segundos entre paquetes para el caso ARP. Para el caso ICMPv6 dependerá del escenario, tal y como se expone a continuación.

Como ya se ha mencionado, se realizará primero sobre una red con 5 nodos y posteriormente con 10. En el apartado 6.4.3 se muestra también el resultado obtenido para una red con 20 nodos que hace uso en el controlador de la aplicación implementada. Para facilitar el envío de paquetes, se utiliza un archivo "awk"que genera un archivo que se carga directamente en Mininet, una vez lanzada la topología, de forma que cada nodo realice

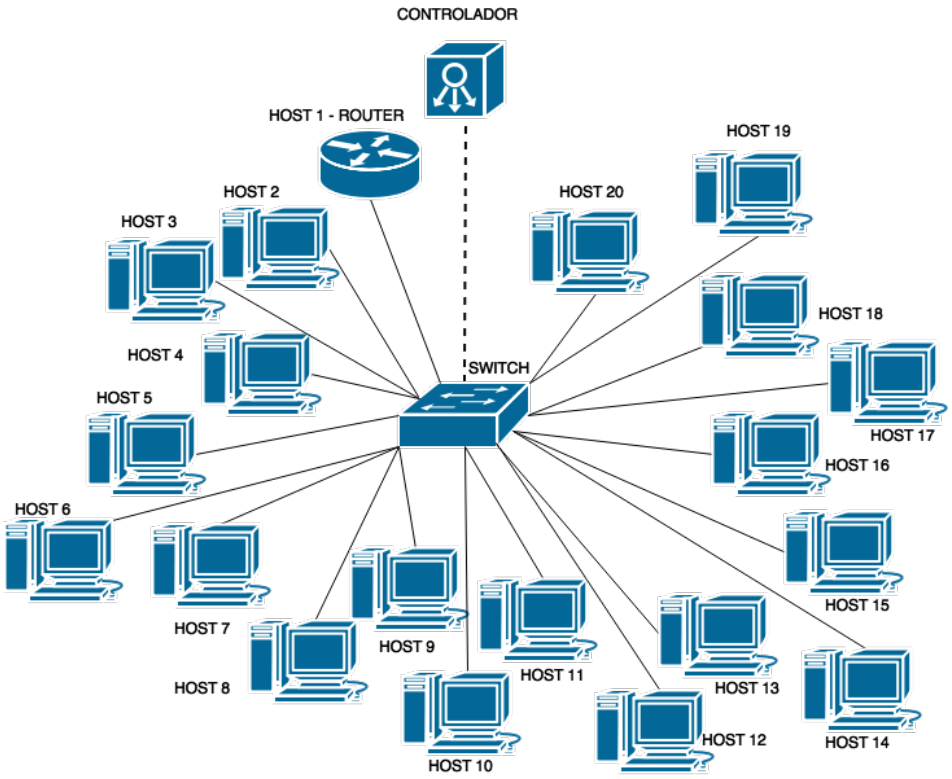


Figura 6.18: Escenario alternativo con 20 nodos y un switch OpenFlow.

simultáneamente ping y ping6 con el resto de nodos de la red. El script implementado para el caso de IPv4 se presenta a continuación:

```
BEGIN{
  n=5; # Número de hosts.
  nPaquetes=20; # Número de paquetes a enviar con ping.
  intervalo=5; # 5 segundos entre paquetes.

  # Para cada origen
  for(i=1;i<=n;i++){
origen="h"i;

# Para cada destino
for(j=1;j<=n;j++){
# Si no es él mismo:
if(i!=j){
destino2="h"j;
print origen" ping -c "nPaquetes" -i
"intervalo" "destino2" &";
```

```
}
        }
  }
}
```

Para el caso de IPv6 se utilizará:

```
BEGIN{
  n=5; # Número de hosts.
  nPaquetes=20; # Número de paquetes a enviar con ping.
  intervalo=5; # 5 segundos entre paquetes.

  # Para cada origen
  for(i=1;i<=n;i++){
origen="h"i;

# Para cada destino
for(j=2;j<=n;j++){
# Si no es él mismo:
if(i!=j){
destino="2001:db8::200:ff:fe00:"j;
print origen" ping6 -c "nPaquetes" -i
"intervalo" "destino" -I "origen"-eth0 &";
}
        }
  }
}
```

Para lanzar estos archivos creando el fichero que posteriormente se cargará, se utiliza, desde el directorio donde se encuentre, el comando:

```
awk -f lanzartodos.awk >  script
```

Esto genera, para el caso de 5 hosts con IPv6 (sería similar para el resto de nodos o para IPv4), un archivo de texto que contiene:

```
h1 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:2 -I h1-eth0 &
h1 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:3 -I h1-eth0 &
h1 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:4 -I h1-eth0 &
h1 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:5 -I h1-eth0 &
h2 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:3 -I h2-eth0 &
h2 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:4 -I h2-eth0 &
h2 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:5 -I h2-eth0 &
```

```
h3 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:2 -I h3-eth0 &
h3 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:4 -I h3-eth0 &
h3 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:5 -I h3-eth0 &
h4 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:2 -I h4-eth0 &
h4 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:3 -I h4-eth0 &
h4 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:5 -I h4-eth0 &
h5 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:2 -I h5-eth0 &
h5 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:3 -I h5-eth0 &
h5 ping6 -c 20 -i 5 2001:db8::200:ff:fe00:4 -I h5-eth0 &
```

Posteriormente, desde Mininet se utiliza "source"para cargar ese fichero:

```
mininet> source script
```

Además, se creó otro archivo para la configuración automática de IPv6, ya que tal y como se especifica en el apéndice A, es necesario escribir en cada uno de los nodos una serie de comandos, por lo que al aumentar los nodos de la red es más compleja la configuración y con un archivo que lo automatiza se agiliza el proceso:

```
BEGIN{
n=5
print "h1 echo 1 > /proc/sys/net/ipv6/conf/all/forwarding";
for(i=2;i<=n;i++){
origen="h"i;
print origen" echo 0 > /proc/sys/net/ipv6/conf/"
origen"-eth0/accept_dad";
}
print "s1 sysctl -w net.ipv6.conf.all.disable_ipv6=0"
print "s1 ip -6 addr add 2001:db8:0:0::1/64 dev s1-eth1"
}
```

Lo cual genera:

```
h1 echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
h2 echo 0 > /proc/sys/net/ipv6/conf/h2-eth0/accept_dad
h3 echo 0 > /proc/sys/net/ipv6/conf/h3-eth0/accept_dad
h4 echo 0 > /proc/sys/net/ipv6/conf/h4-eth0/accept_dad
h5 echo 0 > /proc/sys/net/ipv6/conf/h5-eth0/accept_dad
h6 echo 0 > /proc/sys/net/ipv6/conf/h6-eth0/accept_dad
h7 echo 0 > /proc/sys/net/ipv6/conf/h7-eth0/accept_dad
h8 echo 0 > /proc/sys/net/ipv6/conf/h8-eth0/accept_dad
h9 echo 0 > /proc/sys/net/ipv6/conf/h9-eth0/accept_dad
```

```
h10 echo 0 > /proc/sys/net/ipv6/conf/h10-eth0/accept_dad
s1 sysctl -w net.ipv6.conf.all.disable_ipv6=0
s1 ip -6 addr add 2001:db8:0:0::1/64 dev s1-eth1
```

Tras esto, se realiza el inicio del servicio "radvdz se comienza a capturar con Wireshark.

Mencionar sin embargo que para el caso de 10 nodos con IPv6, ya que la transmisión de 20 paquetes por nodo y por destino, junto con el intervalo entre paquetes suponían un elevado tiempo de ejecución, se optó por otra prueba similar. En lugar de 20 paquetes se transmitía 1. Posteriormente, con el script que se presenta a continuación se genera un código para deshabilitar todas las interfaces del sistema:

```
BEGIN{
n=10
for(i=2;i<=n;i++){
origen="h"i;
print "link s1 "origen" down";
}
}
```

Y con el mismo código, cambiando *down* por *up* se habilitan posteriormente. Tras esto, se vuelve a lanzar el código para enviar un paquete ping por nodo a cada uno de los nodos pertenecientes a la red, y una vez finalizado se extraen las estadísticas de Wireshark.

6.4.3. Resultados obtenidos

A continuación, se presentan los resultados obtenidos para cada una de las redes en los dos sistemas, sin la aplicación y con la aplicación, con la intención de analizar si la solución implementada supone o no una mejora para la red.

En primer lugar, se realizó sobre una red con 5 nodos. Las estadísticas obtenidas para el caso de ARP muestran que en el caso de usar el controlador sin la solución implementada se envían 94 paquetes de este protocolo, frente a los 30 que se envían en el controlador que utiliza la solución. Se puede afirmar por tanto que la solución filtra este tráfico satisfactoriamente, reduciéndolo a la tercera parte del tráfico generado sin solución.

Los resultados para la misma red en el caso de IPv6 muestran que el número de mensajes enviados es muy superior, aunque de nuevo se observa que en el caso sin solución se envían más paquetes ICMPv6. En concreto, se envían 1064 frente a los 654 del caso con solución, por lo que se concluye

que en esta red con 5 nodos también supone un filtrado satisfactorio para IPv6.

Del mismo modo, se observa un comportamiento similar para el caso de ARP, en el que el controlador sin solución transmite 1585 y el caso con solución 811. De nuevo se aprecia una clara diferencia, si bien es cierto que la proporción es menor en este caso.

Para el caso de IPv6, como ya se ha comentado, se realizó una variante en esta prueba, por lo que el número total de paquetes enviados en el sistema y el tiempo empleado es diferente que para IPv4. En consecuencia, se obtiene que para el caso sin solución se han transmitido 9348 y para el caso con solución 7947. De haberse realizado una prueba transmitiendo el mismo número de paquetes que para IPv4, el número de paquetes transmitidos en ambos casos habría sido bastante superior, aunque en este mismo experimento se puede observar cómo el caso que implementa la solución mantiene un tráfico inferior al caso sin solución a pesar del aumento de nodos en la red, si bien la diferencia entre ambos ha disminuido conforme ha aumentado el número de nodos.

Por último, se intentó aumentar el número de nodos para realizar una última prueba con 20 hosts, uno de ellos funcionando como router. Para el caso que no aplicaba la solución fue imposible completar esta prueba, ya que pasado un tiempo el controlador se bloqueaba y el sistema completo colapsaba, por lo que tras varios intentos se desistió. Sin embargo, para el caso que aplicaba la solución sí que fue posible completar esta prueba, en menos tiempo que para la red de 10 nodos y con un tráfico de casi cinco veces dicho sistema.

Se presenta para concluir la siguiente tabla con el resumen de las estadísticas para cada caso estudiado, presentando el número de paquetes transmitidos por protocolo en cada caso, así como las gráficas correspondientes a los resultados obtenidos para 5 y 10 nodos, ya que en el caso de 20 nodos no es posible la comparación.

	ARP		**ICMPv6**	
	Sin Filtrado	Con Filtrado	Sin Filtrado	Con Filtrado
5 Nodos	94	30	1064	654
10 Nodos	1585	811	9348	7947
20 Nodos	-	63458	-	583506

Tabla 6.1: Número de paquetes transmitidos para cada caso en la prueba de rendimiento.

Debido a las limitaciones en cuanto a equipamiento para la realización del proyecto, los experimentos fueron realizados para redes con un máximo de 20 nodos. Podría considerarse insuficiente, ya que la disminución del

rendimiento del sistema que se planteó como antecedente y motivación del proyecto no era notable hasta experimentar en redes con 1000 nodos. Por tanto, el estudio del rendimiento en IPv6 debería continuarse para redes de mayor volumen.

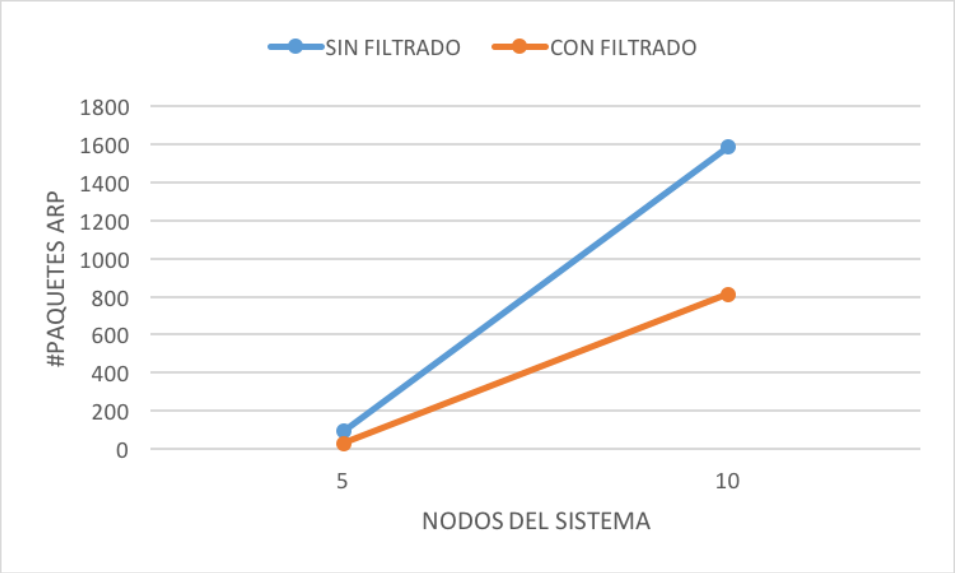


Figura 6.19: Comparación de las estadísticas obtenidas para el tráfico ARP.

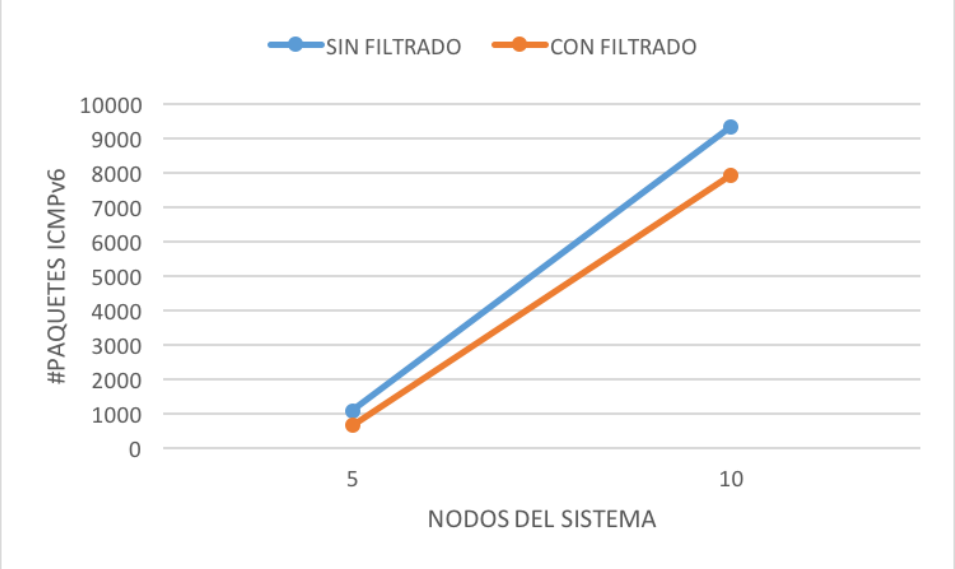


Figura 6.20: Comparación de las estadísticas obtenidas para el tráfico ICMPv6.

6.4.4. Conclusiones

Se presenta a continuación un resumen de los resultados obtenidos a modo de conclusión. En primer lugar, tal y como se observa en la tabla **??**, la disminución de paquetes cuando se aplica filtrado es notable. Concretamente, hay un 70 % de mejora para el caso con 5 nodos y un 50 % para el caso de 10 nodos. El caso de 20 nodos ni siquiera se pudo comparar, ya que el sistema sin filtrado se bloqueaba por completo debido a la gran cantidad de paquetes enviados y retransmitidos.

Por otra parte, respecto a ICMPv6, se observa para el caso de 5 nodos con filtrado una mejora del 40 %, mientras que para el caso de 10 nodos la mejora se limita al 15 %. Si bien es cierto que el número de paquetes filtrados en ICMPv6 es superior, la cantidad de tráfico generado por este protocolo es tal que el porcentaje total de mejora es inferior al de ARP.

Capítulo 7

Conclusiones y trabajo futuro

Este último capítulo recoge las conclusiones finales tras la realización del Trabajo Fin de Grado, así como la proyección de futuro del mismo, presentando algunas vías de investigación planteadas. Por último, se presenta una valoración personal del proyecto en sí, así como de las tecnologías implicadas en el mismo.

7.1. Conclusiones

En el presente trabajo se ha diseñado e implementado el soporte para algunos servicios de autoconfiguración y resolución de direcciones en una red de área local extensa mediante el uso de Software Defined Networking. Para ello, tras una revisión del estado del arte de estas redes y de las tecnologías asociadas a las mismas, se optó por implementar una aplicación dentro del controlador elegido, OpenDayLight, que permitiera realizar una identificación de paquetes, seguida por un filtrado y reenvío. Consecuentemente, se reduciría el tráfico en situaciones de *broadcast* o inundaciones en la red, aumentando el rendimiento del sistema.

Destaca, como una de las contribuciones principales de este proyecto, la investigación realizada sobre SDN y las herramientas asociadas, ya que se considera una de las posibilidades para afrontar las limitaciones que las redes actuales suponen en el avance de las telecomunicaciones. Entre dichas limitaciones se puede encontrar la disminución del rendimiento del sistema que suponen las tormentas de *broadcast*, lo cual queda resuelto gracias al desarrollo del filtro del controlador, contribución esencial del proyecto. Cabe mencionar también la implementación para IPv6, actualmente en auge y en proceso de sustituir por completo al protocolo IPv4 en un futuro próximo.

Por último, el diseño escalable de la implementación ha permitido adaptar al protocolo ARP la solución inicial, lo cual puede dar pie en un futuro a implementar otros protocolos que reduzcan el rendimiento de la red, avanzando hacia la optimización de las redes de comunicación.

Se ha realizado por tanto el diseño de un algoritmo de filtrado para los protocolos ARP e ICMPv6. Posteriormente, se ha implementado en una red SDN con un controlador ODL emulada mediante el uso de la herramienta Mininet, y finalmente se ha evaluado en diferentes escenarios.

Respecto a los resultados obtenidos, evaluados mediante diferentes pruebas, muestran que la solución diseñada funciona correctamente de acuerdo a lo esperado. En concreto, la reducción del tráfico ARP es notable, y tal y como se muestra la mejora de rendimiento del sistema puede considerarse satisfactoria.

7.2. Proyección de futuro

Aunque se considera que los objetivos de este proyecto se han cumplido satisfactoriamente, queda planteada la posibilidad de continuarlo, logrando una implementación más completa y competitiva que, de ser posible, mejore los resultados obtenidos. Especialmente, para el caso de IPv6. Si bien funciona y aumenta el rendimiento del sistema reduciendo el tráfico del mismo, podría mejorarse filtrando aún más los paquetes ICMPv6, de forma que la red sea totalmente escalable.

Por un lado, las redes SDN se encuentran en continuo desarrollo, así como el controlador ODL, por lo que sería interesante continuar con el estudio teórico de estas tecnologías emergentes. Además, la integración de la solución en una red real podría facilitar la evolución de las comunicaciones basándose sobre dichas tecnologías, así como comprobar la mejora real que la solución implementada supone.

Respecto a la solución, haber realizado un diseño escalable da pie a la extensión de la misma a otros protocolos, además de abrir la posibilidad de mejorar el diseño de IPv6 en caso de que fuera necesario. ICMPv6 y ARP se consideran los protocolos que más tráfico generan, lo que justifica su elección a la hora de realizar la implementación. Sin embargo, sería interesante continuar con protocolos como DHCP, Bonjour, etc.

7.2.1. Arquitectura 5G

Inicialmente, se considera la posibilidad de diseñar una arquitectura para la próxima generación móvil basada en una red Ethernet implementada sobre SDN. Entre los retos planteados, se incluye la escalabilidad del sistema

y la necesidad de un rendimiento óptimo que agilice las comunicaciones.

Utilizar SDN para realizar la separación del plano de control y de datos del sistema, recurriendo a un controlador implementable como ODL que podría integrarse fácilmente en la red, posibilita la consecución de los objetivos planteados y presenta nuevas vías de investigación en la evolución de las redes de comunicación.

7.3. Valoración personal

Para concluir, se presenta una evaluación personal del desarrollo de este Trabajo Fin de Grado.

En primer lugar, respecto a las tecnologías implicadas en el mismo. Si bien es cierto que SDN se presenta como una revolución tecnológica que avanza a gran velocidad, la documentación disponible de la misma para el desarrollo de cualquier servicio es limitada y el encontrarse en continua evolución supone una de las mayores dificultades afrontadas en la consecución de este trabajo.

Por otra parte, este trabajo representa la consecución final de las aptitudes necesarias para la obtención del título de Grado, por lo que pone de manifiesto las capacidades adquiridas en el transcurso del mismo. Se podría decir que el objetivo final es adquirir conocimientos sobre un área de estudio específica y aplicar dichos conocimientos para reunir, analizar e interpretar datos relevantes en una investigación, y posteriormente ser capaces de presentar dichos datos a un público, especializado o no en ese área. La intención de esta memoria es documentar el proceso seguido para lograr dicho objetivo, así como el resto de objetivos planteados, que han sido alcanzados satisfactoriamente.

Bibliografía

[1] A. F. Cattoni, P. E. Mogensen, S. Vesterinen, M. Laitila, L. Schumacher, P. Ameigeiras, and J. J. Ramos-Munoz, "Ethernet-based mobility architecture for 5g," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014.

[2] J. Wang, W. Zhao, S. Yang, J. Liu, T. Huang, and Y. Liu, "Fsdm: Floodless service discovery model based on software-defined network," in *2013 IEEE International Conference on Communications Workshops (ICC)*, June 2013, pp. 230–234.

[3] Cisco, "Internetwork design guide - broadcasts in switched lan internetworks." [Online]. Available: http://docwiki.cisco.com/wiki/Internetwork_Design_Guide_--_Broadcasts_in_Switched_LAN_Internetworks#Table:_Average_Number_of_Broadcasts_and_Multicasts_for_Novell_Networks

[4] ——, "Visual networking index: Global mobile data traffic forecast update, 2015–2020," *Tech. Rep., Cisco*, 2016. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[5] O. N. Foundation, "Software-defined networking: The new norm for network," Tech. Rep., April 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[6] *IEEE Standard for Ethernet.*, Revision of ieee std 802.3-2008 ed., IEEE Computer Society, N.Y., December 2012.

[7] P. Ameigeiras, J. J. Ramos-Muñoz, L. Schumacher, J. Prados-Garzon, J. Navarro-Ortiz, and J. M. Lopez-Soler, "Link-level access cloud architecture design based on sdn for 5g networks," *IEEE Network*, vol. 29, no. 2, March 2015.

[8] D. C. Plummer, *RFC 826 An Ethernet Address Resolution Protocol*, November 1982.

[9] R. Droms, *RFC 2131 Dynamic Host Configuration Protocol*, March 1997. [Online]. Available: https://www.ietf.org/rfc/rfc2131.txt

[10] S. Thomson, T. Narten, and T. Jinmei, *RFC 4862 IPv6 Stateless Address Autoconfiguration*, September 2007. [Online]. Available: https://tools.ietf.org/pdf/rfc4862.pdf

[11] L. Foundation, *OpenDaylight Developer Guide*, 2015. [Online]. Available: http://go.linuxfoundation.org/l/6342/2015-06-28/2l76qr/6342/128124/bk_developers_guide_20150629.pdf

[12] S. Deering and R. Hinden, *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*, December 1998. [Online]. Available: http://tools.ietf.org/html/rfc2460

[13] B. Lantz, N. Handigol, B. Heller, and V. Jeyakumar, *Introduction to Mininet*, December 2015. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[14] N. R. Organization, "Less than 10 % of ipv4 addresses remain unallocated," July 2010. [Online]. Available: http://web.archive.org/web/20100707103659/https://www.nro.net/media/less-than-10-percent-ipv4-addresses-remain-unallocated.html

[15] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: Challenges and research opportunities for future internet," *Computer Networks*, vol. 75, Part A, pp. 453 – 471, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614003703

[16] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79 – 95, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615000614

[17] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74 – 98, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614002588

[18] F. Longo, S. Distefano, D. Bruneo, and M. Scarpa, "Dependability modeling of software defined networking," *Computer Networks*, vol. 83, pp. 280 – 296, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615001139

[19] Google, "Google openflow," 2013. [Online]. Available: http://opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf

[20] S. Elby, "Verizon - adoption of sdn: Progress update," 2012. [Online]. Available: http://opennetsummit.org/archives/apr12/elby-tue-sdn.pdf

[21] J. Chawki, "How orange is using opendaylight," 2015. [Online]. Available: https://www.opendaylight.org/news/user-story/2015/06/how-orange-using-opendaylight

[22] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1 – 30, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614002254

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[24] M. Smith, R. E. Adams, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, *OpFlex Control Protocol*, Cisco, April 2016. [Online]. Available: https://tools.ietf.org/pdf/draft-smith-opflex-03.pdf

[25] T. N. D. and K. Gray, *SDN: Software Defined Networks*, 1st ed. O'Reilly Media, Inc., 2013.

[26] O. N. Foundation, "Openflow switch specification," Open Networking Foundation, Tech. Rep., 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

[27] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici, "A comparison between several software defined networking controllers," in *Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS), 2015 12th International Conference on*, Oct 2015, pp. 223–226.

[28] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, June 2014, pp. 1–6.

[29] Kanika, "Difference between ad-sal and md-sal." [Online]. Available: http://sdntutorials.com/difference-between-ad-sal-and-md-sal/

[30] E. Rojas, "Clash of titans in sdn: Opendaylight vs onos," Madrid, March 2016. [Online]. Available: http://www.slideshare.net/opennebula/clash-of-titans-in-sdn-opendaylight-vs-onos-elisa-rojas

[31] "Eclipse." [Online]. Available: https://eclipse.org

[32] "Apache maven project." [Online]. Available: https://maven.apache.org

[33] "Mininet: An instant virtual network on your laptop (or other pc)." [Online]. Available: http://mininet.org

[34] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, June 2014, pp. 1–6.

[35] *RFC 791 Internet Protocol*, Information Sciences Institure University of Southern California, September 1981.

[36] K. Elmeleegy and A. L. Cox, "Etherproxy: Scaling ethernet by suppressing broadcast traffic," in *INFOCOM 2009, IEEE*, April 2009, pp. 1584–1592.

[37] A. Conta and S. Deering, *RFC 2463 ICMP for the Internet Protocol Version 6 (IPv6)*, December 1998. [Online]. Available: http://tools.ietf.org/pdf/rfc2463.pdf

[38] M. Casado, Ed., *Ethane: Taking Control of the Enterprise*, vol. 37, no. 4. NY, USA: Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications, October 2007.

[39] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: A scalable ethernet architecture for large enterprises," vol. 29, no. 1. NY, USA: ACM Trans. Computer Systems (TOCS), February 2011.

[40] T. Narten, E. Nordmark, and W. Simpson, *RFC 2461 Neighbor Discovery for IP Version 6 (IPv6)*, December 1998. [Online]. Available: https://tools.ietf.org/pdf/rfc2461.pdf

[41] P. L. Weiguo, "Address resolution protocol," September 2015. [Online]. Available: http://icourse.cuc.edu.cn/networkprogramming/lectures/Unit2_ARP.pdf

[42] O. N. Foundation, "Software-defined networking (sdn) definition," May 2013. [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition

[43] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the sdn control traffic in large isp networks," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5248–5253.

[44] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of opendaylight sdn controller," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2014, pp. 671–676.

[45] "Tutorial sdn." [Online]. Available: http://sdnhub.org/tutorials/sdn-tutorial-vm/

Apéndice A

Instalación y puesta en marcha de las herramientas

En este apéndice se recoge la documentación necesaria para la configuración y utilización de las herramientas necesarias para la realización de este proyecto.

A.1. Emulador y controlador

En primer lugar, será necesario instalar el emulador y añadir el controlador y las herramientas de programación y compilación mencionadas en la sección 4.2.2.

Como se comentó inicialmente, la forma más sencilla de operar con el emulador es mediante el uso de una máquina virtual, aunque también existe la posibilidad de instalarlo de forma nativa. Con una máquina virtual se asegura que la ejecución del mismo no afecte al resto del equipo, en caso de que haya algún error o que sea necesario modificar la configuración de la red.

Como recomendación, la imagen disponible en la página de SDN [45] dispone de todas las herramientas mencionadas ya instaladas, así como de un tutorial para familiarizarse con OpenDayLight. Dicha imagen dispone de la última versión disponible de Mininet, aunque la versión puede comprobarse directamente con el comando:

```
mn --version
```

En caso de ser necesario, se puede actualizar tal y como indica la página del propio emulador. [33] Lo que sí es necesario instalar y actualizar cada vez

que se inicie el sistema o se realice algún cambio en el código del controlador será Maven. Para actualizarlo y lanzar el controlador, desde el terminal:

```
ubuntu@sdnhubvm:~$ cd SDNHub_Opendaylight_Tutorial
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial$ mvn clean install -nsu
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial$ cd
distribution/opendaylight-karaf/target/assembly
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial/distribution/
opendaylight-karaf/target/assembly$ ./bin/karaf
```

En el mismo controlador, una vez iniciado, será necesario instalar las aplicaciones del controlador de las que se desea hacer uso. Se puede comprobar cuáles se encuentran activas con el comando:

```
opendaylight-user@root> feature:list | grep sdnhub
```

El emulador está capacitado para soportar multitud de topologías diferentes, desde programadas con Python hasta las más sencillas lanzadas con una línea de comandos desde el mismo terminal. Para que considere el controlador será necesario indicarlo. La topología más sencilla, utilizada en la fase de implementación de este proyecto, se lanzaría de la siguiente forma:

```
ubuntu@sdnhubvm:~$ sudo mn --topo single,3 --mac --switch
 ovsk,protocols=OpenFlow13 --controller remote
```

Y una vez hecho esto, será necesario instalar el flujo que conecte con el controlador para posibilitar la comunicación entre nodos:

```
s1 ovs-ofctl add-flow tcp:127.0.0.1:6634 -OOpenFlow13
 priority=1,action=output:controller
```

Como ya se ha mencionado, es necesario actualizar Maven desde el directorio en el que se encuentra cada vez que se edite el código del controlador, ya que existe un archivo "pom.xml"que contiene toda la información referente a las dependencias del proyecto programado y es indispensable para la compilación del mismo.

Por último, mencionar que tanto Mininet como OpenDayLight ofrecen tutoriales para facilitar la familiarización de los usuarios, que contienen los comandos disponibles en cada una de las herramientas para un mejor aprovechamiento de las posibilidades que ofrecen.

A.2. IPv6

Para la correcta implementación en IPv6 será necesario realizar la configuración de este protocolo en el emulador. Como primeras consideraciones, Mininet incluye desde 2014 la posibilidad de utilizar IPv4 e IPv6 conjuntamente, aunque tan sólo asigna a sus nodos una dirección de enlace, por lo que para realizar la asignación global será necesario hacer uso de un router o, en su defecto, un host que actúe como router. Además, para realizar un ping en el caso de IPv6 será necesario utilizar la dirección completa del host destino, y no sólo su identificador, seguido de la interfaz del nodo origen, utilizada para dicha comunicación, por ejemplo si se quisiera realizar un ping de h1 a h2 (utilizando la dirección de enlace de éste último):

```
mininet> h1 ping6 fe80::200:ff:fe00:2 -I h1-eth0
```

Para la configuración de este protocolo en el emulador, será necesario seguir los siguientes pasos en el orden en el que se presentan:

1. Desde el terminal del sistema, instalación del demonio Router Advertisement Daemon (RADVD) mediante el comando:

   ```
   sudo aptitude install radvd
   ```

   Después de la primera instalación no existe el archivo de configuración, por lo que es necesario crearlo en /etc/radvd.conf. Para este proyecto, se usó el siguiente archivo:

   ```
   interface h1-eth0
   {
      AdvSendAdvert on;
      MinRtrAdvInterval 3;
      MaxRtrAdvInterval 10;
      prefix 2001:db8::/64
      {
   AdvOnLink on;
   AdvAutonomous on;
   AdvRouterAddr on;
      };
   };
   ```

   La interfaz será el enlace por el que se transmiten los mensajes de Router Advertisement, dado que en el proyecto se configura siempre

el nodo h1 como router, se establece por defecto la interfaz h1-eth0. Además, se utiliza el prefijo 2001:db8::/64 ya que es un prefijo reservado para documentación.

2. Se lanza la topología y se establecen los flujos de datos. Posteriormente se lanza un terminal por nodo mediante el comando:

```
mininet> xterm <nodo1> <nodo2> <...>
```

3. En los terminales de los diferentes nodos, siendo X el número identificador de cada nodo:

```
ubuntu@sdnhubvm:~$ echo \0" > /proc/sys/net/ipv6/conf/
all/forwarding
ubuntu@sdnhubvm:~$ echo \1" > /proc/sys/net/ipv6/conf/
all/autoconf
ubuntu@sdnhubvm:~$ echo \1" > /proc/sys/net/ipv6/conf/
all/accept_ra
ubuntu@sdnhubvm:~$ echo \0" > /proc/sys/net/ipv6/conf/
hX-eth0/accept_dad
```

4. En el terminal de Mininet:

```
mininet> s1 sysctl -w net.ipv6.conf.all.disable_ipv6=0
mininet> s1 ip -6 addr add 2001:db8:0:0::1/64 dev s1-eth1
mininet> s1 ifconfig s1-eth1 inet6 add 2001:db8:0:0::1/64
```

5. En el terminal del router:

```
ubuntu@sdnhubvm:~$ echo \1" > /proc/sys/net/ipv6/conf/
all/forwarding
ubuntu@sdnhubvm:~$ echo \1" > /proc/sys/net/ipv6/conf/
all/autoconf
ubuntu@sdnhubvm:~$ echo \1" > /proc/sys/net/ipv6/conf/
all/accept_ra
ubuntu@sdnhubvm:~$ sudo service radvd restart
```

Llegados a este punto, será posible realizar ping6 entre los elementos de la red. Además, al comprobar las direcciones asignadas, se mostrará la de enlace únicamente en el caso del router, y ambas en el resto de los nodos, tal y como muestran las capturas a continuación.

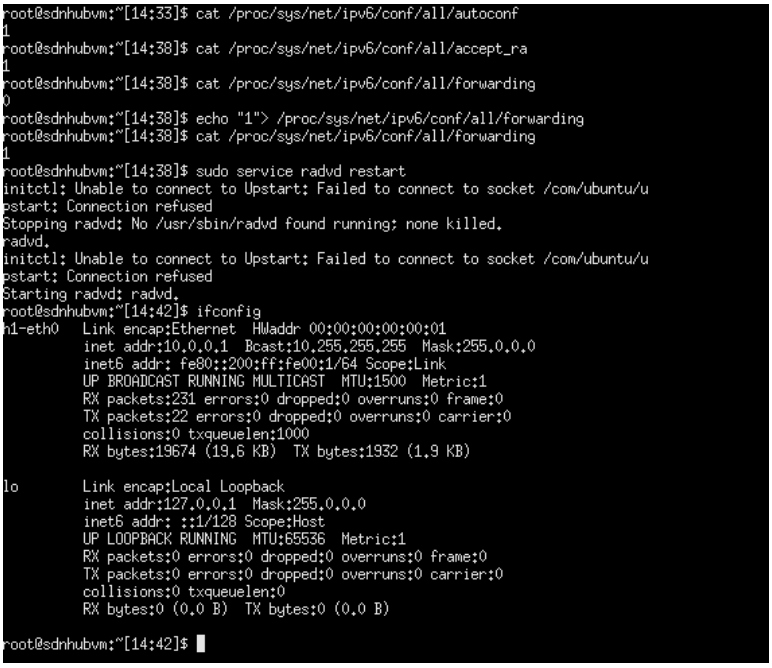


Figura A.1: Captura configuración host actuando de router.

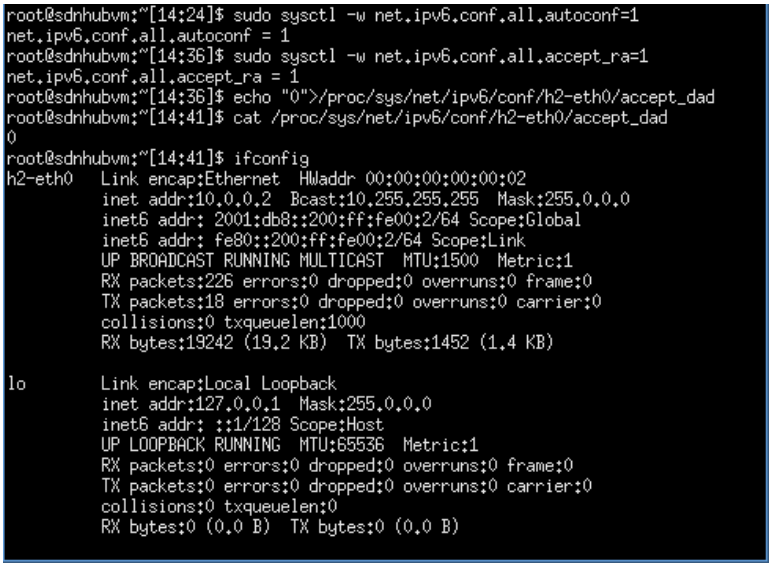


Figura A.2: Captura configuración host.

```
root@sdnhubvm:~[14:37]$ cat /proc/sys/net/ipv6/conf/all/autoconf
1
root@sdnhubvm:~[14:37]$ cat /proc/sys/net/ipv6/conf/all/accept_ra
1
root@sdnhubvm:~[14:37]$ cat /proc/sys/net/ipv6/conf/all/forwarding
0
root@sdnhubvm:~[14:37]$ cat /proc/sys/net/ipv6/conf/h3-eth0/accept_dad
1
root@sdnhubvm:~[14:39]$ echo "0"> /proc/sys/net/ipv6/conf/h3-eth0/accept_dad
root@sdnhubvm:~[14:39]$ cat /proc/sys/net/ipv6/conf/h3-eth0/accept_dad
0
root@sdnhubvm:~[14:40]$ ifconfig
h3-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:03
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: 2001:db8::200:ff:fe00:3/64 Scope:Global
          inet6 addr: fe80::200:ff:fe00:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:224 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19037 (19.0 KB)  TX bytes:816 (816.0 B)
```

Figura A.3: Captura configuración alternativa.

```
ubuntu@sdnhubvm:~[14:18]$ sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow13 --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> s1 ovs-ofctl add-flow tcp:127.0.0.1:6634 -OOpenFlow13 priority=1,action=output:controller
mininet> xterm h1 h2 h3
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2196 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1392 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.362 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1060 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.135 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5016ms
rtt min/avg/max/mdev = 0.098/775.138/2196.876/845.157 ms, pipe 3
mininet> s1 sysctl -w net.ipv6.conf.all.disable_ipv6=0
net.ipv6.conf.all.disable_ipv6 = 0
mininet> s1 ip -6 addr add 2001:db8:0:0::1/64 dev s1-eth1
mininet> s1 ifconfig s1-eth1 inet6 add 2001:db8:0:0::1/64
SIOCSIFADDR: File exists
mininet>
```

Figura A.4: Captura configuración mininet.

Apéndice B

Problemas frecuentes

A continuación se presentan algunos de los problemas afrontados a lo largo de la resolución del trabajo, principalmente debidos a errores en la configuración de las herramientas.

- **Compatibilidad del emulador.** Como ya se ha comentado, el sistema operativo ideal para trabajar con Mininet es Ubuntu. Lógicamente, con una máquina virtual el funcionamiento y el rendimiento del emulador no será el mismo que de realizarse una instalación nativa en el sistema. Otra posibilidad es arrancar el sistema desde una memoria externa con una versión *alive* de Ubuntu, preferiblemente con Mininet instalado. Sin embargo, esta opción no permite guardar los cambios realizados en el controlador ni los programas instalados para la compilación, por lo que, de no disponer de un equipo con Ubuntu instalado, la opción más razonable será utilizar una máquina virtual con la imagen disponible en [33].

- **Mininet no se conecta a OpenDayLight.** Tras lanzar Mininet indicando OpenDayLight como controlador remoto e instalar los flujos necesarios, no es posible realizar ping entre los nodos de la red. Esto puede deberse principalmente a dos motivos:

  - El controlador no se ha iniciado por completo. Ocasionalmente, el controlador necesita más tiempo del usual para iniciarse y permitir la comunicación entre los nodos. Una forma de comprobarlo es al lanzar Mininet: cuando añade al controlador aparece un mensaje anunciando la conexión con el mismo. De no haber pasado un margen de tiempo suficiente, anunciará que ha sido imposible conectarse, aunque dicha conexión se realizará automáticamente cuando el controlador esté disponible.

- Maven no se ha instalado correctamente. La página que presenta el tutorial de SDN [45] indica que la instalación de Maven debe realizarse de la siguiente forma:

  ```
  ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial$ mvn
  install -nsu
  ```

  Sin embargo, una instalación limpia resulta óptima y la diferencia de tiempo necesario para realizarla es ligeramente superior comparada con la instalación normal.

  ```
  ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial$ mvn clean
  install -nsu
  ```

- **El controlador presenta errores ajenos al código implementado.** El código proporcionado en la imagen instalada en la máquina virtual tiene errores por defecto en el archivo "pom.xml". Será necesario repararlo, ya que se utilizará el mismo archivo en cualquier implementación realizada. Para ello, es necesario actualizar las librerías desde el mismo archivo, así como instalar las extensiones ausentes.

- **El código del controlador no se actualiza.** Cada vez que se realice un cambio en el código del controlador es necesario parar el controlador en caso de que se esté ejecutando, volver a instalar Maven y reiniciar el controlador. Adicionalmente, se debe actualizar el proyecto desde la misma ventana de Eclipse, en las opciones específicas de Maven.

- **IPv6 no realiza correctamente la asignación de direcciones.** Si se han seguido las instrucciones del apéndice A y aún así no es posible configurar correctamente este protocolo, habrá que añadir en el fichero sysctl.conf la siguiente línea:

  ```
  net.ipv6.conf.all.use_tempaddr=-1
  ```