



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

– **TELECOM** ESCUELA
TÉCNICA **VLC** SUPERIOR
DE INGENIERÍA DE
TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de
Telecomunicación

Estudio de una red SDN mediante el switch virtual Open
vSwitch y el controlador Ryu.

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de
Telecomunicación

AUTOR/A: Sellés Llinares, Sergi

Tutor/a: Romero Martínez, José Oscar

CURSO ACADÉMICO: 2024/2025



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

— **TELECOM** ESCUELA
TÉCNICA **VLC** SUPERIOR
DE INGENIERÍA DE
TELECOMUNICACIÓN

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/
CAMPUS
VALENCIA, INTERNATIONAL
CAMPUS OF EXCELLENCE



Resumen

A raíz de la creciente demanda de servicios de Internet, así como el coste y la escalabilidad de las redes, ha llevado a que éstas estén llegando a su límite. Debido a esto surgió la idea una nueva arquitectura de red conocida como SDN. Gracias a las redes definidas por software, la red actual puede ser modificada para que sea mucho más flexible, escalable y reutilizable permitiendo reducir el tiempo de realización de cambios en la red. A causa de esto se reducirían los costes, también la creación de nuevos servicios de una forma más rápida.

Con este trabajo se pretende estudiar el funcionamiento de una SDN mediante el uso de un switch virtual de código abierto conocido como Open vSwitch el cual instalaremos en una máquina virtual con un sistema basado en Linux. Además, se dispondrá del controlador SDN Ryu para gestionar el funcionamiento de la red mediante diferentes componentes de software y de interfaces API. La simulación de la red se realizará conectando varias Raspberry Pi a nuestro switch virtual, de forma práctica simularemos distintas situaciones de la red evaluando los resultados obtenidos.

Resum

Com a resultat de la creixent demanda de serveis d'Internet, així com el cost i l'escalabilitat de les xarxes, ha portat al fet que aquestes estiguen arribant al seu límit. Degut a açò va sorgir la idea d'una nova arquitectura de xarxa coneguda com SDN. Gràcies a les xarxes definides per software, la xarxa actual pot ser modificada perquè siga molt més flexible, escalable i reutilitzable, permetent reduir el temps de realització de canvis en la xarxa. A causa d'això es reduirien els costos, també la creació de nous serveis d'una forma més ràpida.

Amb aquest treball es pretén estudiar el funcionament d'una SDN mitjançant l'ús d'un switch virtual de codi obert conegut com Open vSwitch el qual instal·larem en una màquina virtual amb un sistema basat en Linux. A més, es disposarà del controlador SDN Ryu per a gestionar el funcionament de la xarxa mitjançant diferents components de software i d'interfícies API. La simulació de la xarxa es realitzarà connectant diverses Raspberry Pi al nostre switch virtual, de manera pràctica simularem diferents situacions de la xarxa avaluant els resultats obtinguts.

Abstract

Nowadays, with an increasing demand for internet related services, as also happens with the price and scalability of the networks, has caused them to reach a critical situation. Because of this, the idea of a new type of net architecture known as SDN was brought up. Thanks to software defined networks, the current network can be modified in order to make it more flexible, scalable and reusable, allowing lower waiting times between modifications to be achieved. Due to those lower times, costs will be reduced.

The aim of this project is to study how the SDN works with the use of an open code virtual switch known as Open vSwitch, which will be installed in a virtual machine running Linux. Also, we will have access to the SDN Ryu controller in order to manage the network through different software components and API interfaces. The network simulation will be carried out by connecting several Raspberry Pi to our virtual switch. Our results will be accomplished in a practical manner, analyzing the effects of the different scenarios that occur to our network.

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la ingeniería de telecomunicación

CONCEPT (ABET)	CONCEPTO (traducción)	¿Cumple? (S/N)	¿Dónde? (páginas)
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	6 – 8
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	20 – 41
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	2
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	42 – 73
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	42 – 73
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	74
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	74



Índice

Capítulo 1.	Introducción	1
1.1	Motivación	2
1.2	Objetivos	2
1.3	Contenidos y Organización:	3
Capítulo 2.	Metodología del proyecto.....	4
Capítulo 3.	Estado del arte de las redes tradicionales	6
3.1	Introducción a las redes tradicionales	6
3.2	Problemática de las redes tradicionales.....	6
3.3	Aplicaciones y necesidades actuales	7
3.4	Perspectivas futuras de las redes tradicionales.....	8
Capítulo 4.	Estudio de las Redes Definidas por Software.....	9
4.1	Fundamentos y arquitectura de las redes SDN.....	9
4.2	Componentes de las redes SDN	11
4.2.1	Controlador SDN.....	11
4.2.2	Switch SDN.....	12
4.2.3	Protocolos y estándares	12
4.3	Entorno de red del proyecto	13
4.3.1	Controlador Ryu.....	13
4.3.2	Switch virtual – Open vSwitch.....	15
4.3.3	OpenFlow	16
4.3.4	Dispositivos de red – Raspberry Pi	16
4.4	Ventajas y desventajas del uso de SDN frente a las redes tradicionales	17
4.4.1	Ventajas y Desventajas de las SDN	17
4.4.2	Ventajas y Desventajas de las Redes Tradicionales	17
4.5	Perspectivas futuras de las redes SDN	19
Capítulo 5.	Estudio y resultados prácticos de una red SDN	20
5.1	Preparación del entorno práctico.....	20
5.1.1	Instalación y configuración de Open vSwitch.....	21
5.1.2	Instalación y configuración del controlador Ryu	23
5.1.3	Configuración de las Raspberry Pi.....	32
5.1.4	Creación del puente virtual br0	39
5.2	Diseño y análisis de distintos casos prácticos	42



5.2.1	Caso 1: Conexión entre los dispositivos – Visibilidad de la red	42
5.2.2	Caso 2: Configuración y Pruebas de Control de Tráfico y Calidad de Servicio (QoS)	47
5.2.3	Caso 3: Configuración y uso de VLANs.....	60
Capítulo 6.	Conclusiones y trabajos futuros	74
Capítulo 7.	Bibliografía.....	76



Índice de figuras

Figura 1. Metodología del proyecto	5
Figura 2. Plano de Control y Plano de Datos de las redes.....	9
Figura 3. Interfaces API [23].....	10
Figura 4. Controlador RYU [34].....	13
Figura 5. Compatibilidad de RYU con switches [34]	14
Figura 6. Open vSwitch logo [29].....	15
Figura 7. Open vSwitch logo [31].....	16
Figura 8. Raspberry Pi logo [38].....	16
Figura 9. Esquema del proyecto.....	20
Figura 10. Creación de la máquina virtual (I)	21
Figura 11. Creación de la máquina virtual (II).....	21
Figura 12. Asistente de instalación Ubuntu 24.0.4 LTS	22
Figura 13. Comandos de actualización de software del sistema	22
Figura 14. Comando de instalación de Open vSwitch	22
Figura 15. Comando para comprobar versión de Open vSwitch.....	23
Figura 16. Versión de Open vSwitch instalada	23
Figura 17. Comando para comprobar versión de Python.....	23
Figura 18. Versión de Python por defecto del sistema.....	23
Figura 19. Comando instalación Pyenv - Python 3.9.0.....	24
Figura 20. Proceso de instalación Python 3.9.0	24
Figura 21. Comando para establecer uso de la versión de Python 3.9.0 en el sistema.....	25
Figura 22. Instalación herramienta get-pip.py.....	25
Figura 23. Proceso de instalación de la herramienta get-pip.py.....	25
Figura 24. Versiones de pip y Python compatibles con Ryu.....	26
Figura 25. Comandos de instalación de Ryu	26
Figura 26. Proceso de instalación de Ryu	26
Figura 27. Comprobar versión de Ryu instalada.....	27
Figura 28. Versión de Ryu instalada en el sistema.....	27
Figura 29. Comprobar versión de Open vSwitch instalada.....	27
Figura 30. Versión de Open vSwitch instalada en el sistema	27
Figura 31. Obtención de información del bridge br0 de Open vSwitch.....	28
Figura 32. Información del bridge br0	28
Figura 33. Comando de ejecución de Ryu	28
Figura 34. Ejecución de ryu-manager simple_switch_13.py	29



Figura 35. Comando de verificación de la ejecución de Ryu en el puerto 6633	30
Figura 36. Proceso ejecutandose en el puerto 6633	30
Figura 37. Asignación de Ryu a Open vSwitch	30
Figura 38. Comprobación del controlador asignado a Open vSwitch.....	30
Figura 39. Asignación correcta del controlador	30
Figura 40. Controlador Ryu asignado a Open vSwitch.....	31
Figura 41. Herramienta Raspberry Pi Imager	32
Figura 42. Instalación del S.O en Raspberry Pi 3 (I)	33
Figura 43. Instalación del S.O en Raspberry Pi 3 (II)	33
Figura 44. Instalación del S.O en Raspberry Pi 3 (III).....	34
Figura 45. Instalación del S.O en Raspberry Pi 3 (IV).....	34
Figura 46. Instalación del S.O en Raspberry Pi 4 (I)	35
Figura 47. Instalación del S.O en Raspberry Pi 4 (II)	35
Figura 48. Instalación del S.O en Raspberry Pi 4 (III).....	36
Figura 49. Instalación del S.O en Raspberry Pi 4 (IV).....	36
Figura 50. Raspberry Pi 3 y 4 conectadas a la red	37
Figura 51. Comando de acceso al archivo interfaces de las Raspberry Pi	37
Figura 52. Configuración IP estática en Raspberry Pi 3	37
Figura 53. Configuración IP estática en Raspberry Pi 4	38
Figura 54. Configuración de red Raspberry Pi 3.....	38
Figura 55. Configuración de red Raspberry Pi 4.....	38
Figura 56. Modificación del archivo interfaces del sistema.....	39
Figura 57. Archivo interfaces modificado.....	39
Figura 58. Comandos para la creación del puente (bridge) br0	40
Figura 59. Verificación del puente br0.....	41
Figura 60. Configuración puente br0	41
Figura 61. Esquema del proyecto.....	42
Figura 62. Creación de las reglas de flujo para conectar todos los dispositivos	43
Figura 63. Ping de comprobación de conexión entre dispositivos	43
Figura 64. Respuesta al ping desde Open vSwitch	44
Figura 65. Respuesta al ping desde Raspberry Pi 3	44
Figura 66. Respuesta al ping desde Raspberry Pi 4	45
Figura 67. Reconexión automática de dispositivo.....	46
Figura 68. Script de Ryu para las pruebas de QoS (I).....	48
Figura 69. Script de Ryu para las pruebas de QoS (II).....	49
Figura 70. Ryu lanzado utilizando el script modificado	50



Figura 71. Comando para consultar las reglas de flujo creadas	50
Figura 72. Reglas de flujo creadas por el script	50
Figura 73. Configuración de la QoS.....	51
Figura 74. Creación de Ingress Policing Rate y Burst	52
Figura 75. Comprobación de QoS y Ingress Policing	52
Figura 76. QoS e Ingress Policing creadas correctamente	53
Figura 77. Interfaces asociadas a QoS e Ingress Policing	53
Figura 78. Configuración interfaz enp0s8.....	54
Figura 79. Configuración interfaz enp0s9.....	54
Figura 80. Comandos de iperf3 para verificar el rendimiento	55
Figura 81. Ejecución de iperf3 en Raspberry Pi 3 (10.0.0.3).....	56
Figura 82. Ejecución de iperf3 en Raspberry Pi 4 (10.0.0.4).....	57
Figura 83. Comandos para reajustar las policing establecidas y comprobar la eliminación de paquetes.....	58
Figura 84. Comando de iperf3 para comprobar la eliminación de paquetes	58
Figura 85. Ejecución del comando anterior para verificar la eliminación de paquetes.....	59
Figura 86. Creación de un puerto troncal en Open vSwitch [57].....	61
Figura 87. Creación de un puerto de acceso en Open vSwitch [57].....	61
Figura 88. Asignación de interfaz enp0s8 a VLAN 1	62
Figura 89. Asignación de interfaz enp0s9 a VLAN 2	63
Figura 90. Entorno de trabajo VLAN 1 y 2 – 2 dispositivos.....	63
Figura 91. simple_switch_13_vlan.py (I)	64
Figura 92. simple_switch_13_vlan.py (II)	65
Figura 93. Reglas de flujo simple_switch_vlan.py	65
Figura 94. Ejecución del script simple_switch_13_vlan.py	66
Figura 95. Log de Ryu tras ejecutar simple_switch_13_vlan.py	66
Figura 96. Tráfico capturado - VLAN ID 1	67
Figura 97. Raspberry Pi con IP 10.0.0.5	68
Figura 98. Adición del nuevo dispositivo al puente virtual br0	68
Figura 99. Entorno de trabajo VLAN 1 y 2 – 3 dispositivos.....	69
Figura 100. Consulta de números de puerto de las interfaces del puente virtual br0.....	69
Figura 101. Reglas de flujo creadas en el script simple_switch_13_vlan.py	70
Figura 102. Ejecución del script simple_switch_13_vlan.py.....	70
Figura 103. Reglas de flujo de simple_switch_13_vlan.py.....	71
Figura 104. Reglas VLAN 1	71
Figura 105. Reglas VLAN 2	71



Figura 106. Regla Table Miss	72
Figura 107. Reglas de Bloqueo	72
Figura 108. Controlador Ryu bloqueando tráfico entre VLANs	73
Figura 109. Tráfico entre dispositivos de una misma VLAN	73



Índice de tablas

Tabla 1. Ventajas y desventajas SDN vs Redes Tradicionales [39].....	18
--	----

Capítulo 1. Introducción

En los últimos años, la cantidad de dispositivos conectados a Internet ha crecido de forma impresionante, y con ello, la demanda de servicios en línea [1]. Este crecimiento ha puesto a prueba las redes tradicionales, que fueron diseñadas cuando las necesidades eran mucho menores. Ahora, estas redes enfrentan problemas para escalar, ser flexibles y manejar los costos operativos. Las redes convencionales, que combinan de forma rígida el plano de control y el plano de datos, no pueden adaptarse rápidamente a las nuevas exigencias del mercado, lo que causa cuellos de botella y limita su eficiencia.

Para solucionar estos problemas, han surgido las Redes Definidas por Software (SDN, por sus siglas en inglés). SDN cambia completamente la forma en que se diseñan y gestionan las redes al separar el plano de control del plano de datos [2]. Esto permite que el control de la red sea programable y centralizado mediante software, mientras que el manejo del tráfico de datos se vuelve más flexible y dinámico. La posibilidad de abstraer la infraestructura de red y controlarla mediante software trae muchos beneficios, como una mayor flexibilidad, mejor escalabilidad, reutilización de recursos y una reducción significativa en los tiempos de implementación y adaptación a los cambios en la red.

Una gran ventaja de SDN es que reduce los costos operativos y de capital [3]. Con una gestión centralizada y automatizada, se necesita menos intervención manual y se optimiza el uso de los recursos de red. Además, facilita la creación y despliegue de nuevos servicios de manera rápida y eficiente, permitiendo responder a las demandas del mercado con mayor agilidad [3]. Esta capacidad de adaptarse y escalar sin incurrir en grandes gastos es vital en un entorno tecnológico que cambia constantemente.

Este proyecto tiene como objetivo estudiar cómo funciona una red definida por software utilizando un switch virtual de código abierto llamado Open vSwitch. Este switch se instalará en una máquina virtual con Linux, creando un entorno controlado para implementar y probar la arquitectura SDN. Open vSwitch es una herramienta poderosa que permite crear y gestionar switches virtuales y soporta protocolos como OpenFlow, que es crucial para la comunicación entre el plano de control y el plano de datos en SDN.

Para gestionar la red, utilizaremos el controlador SDN Ryu, una plataforma que facilita la gestión y orquestación de los flujos de datos a través de diferentes componentes de software y APIs. Ryu ofrece una interfaz programable que permite a los administradores definir políticas de red y gestionar el comportamiento de los dispositivos de manera centralizada y coherente.

Simularemos la red conectando varias Raspberry Pi al switch virtual. Estas Raspberry Pi funcionarán como dispositivos finales en la red, permitiendo simular diferentes situaciones y escenarios de red. Esta configuración práctica nos permitirá evaluar el rendimiento, flexibilidad y escalabilidad de la red SDN en condiciones controladas y realistas.

Con estas simulaciones, esperamos demostrar cómo la arquitectura SDN, combinada con herramientas como Open vSwitch y el controlador Ryu, puede transformar la gestión y operación de redes, ofreciendo soluciones más eficientes y adaptables. Evaluaremos diversos escenarios de red y analizaremos los resultados para identificar las ventajas y posibles limitaciones de implementar SDN en comparación con las redes tradicionales.

En resumen, este trabajo no solo busca proporcionar una comprensión teórica de las Redes Definidas por Software, sino también ofrecer una evaluación práctica de su implementación y funcionamiento. A través de este estudio, esperamos resaltar el potencial de SDN para resolver los desafíos actuales de las redes, mejorando su flexibilidad, escalabilidad y eficiencia operativa.

1.1 Motivación

La revolución digital y el auge del Internet de las cosas (IoT) han provocado un aumento exponencial en la demanda de servicios de Internet. La proliferación de dispositivos conectados, desde teléfonos inteligentes hasta electrodomésticos inteligentes, ha llevado a un crecimiento sin precedentes en el tráfico de datos y la necesidad de redes más robustas y flexibles. Las arquitecturas de red tradicionales, con su rigidez inherente y falta de escalabilidad, están luchando por mantenerse al día con estas crecientes exigencias. Estas redes, diseñadas en una época en la que las demandas eran considerablemente menores, ahora se enfrentan a problemas significativos en términos de capacidad, gestión y costos.

En mi experiencia personal y académica, he observado de primera mano cómo las limitaciones de las redes tradicionales afectan la capacidad de las organizaciones para innovar y adaptarse a nuevas demandas. La necesidad de redes que puedan adaptarse rápidamente a nuevas demandas, escalar eficientemente y reducir costos operativos me ha llevado a explorar nuevas tecnologías y paradigmas. En este contexto, las Redes Definidas por Software (SDN) me han parecido una solución innovadora y fascinante. Las SDN representan un cambio fundamental en el diseño y gestión de redes, separando el plano de control del plano de datos. Esta separación permite que el control de la red sea programable y centralizado a través de software, mientras que el encaminamiento de los datos se gestiona de manera más flexible y dinámica.

Una de las principales ventajas de SDN que me ha atraído es su capacidad para ofrecer una mayor flexibilidad y capacidad de gestión centralizada. Al permitir una gestión centralizada y automatizada, SDN disminuye la necesidad de intervención manual y optimiza el uso de los recursos de red. Esta capacidad de reconfiguración dinámica de la red y de implementación rápida de nuevos servicios es crucial en un entorno tecnológico en constante evolución. Además, SDN facilita la creación y el despliegue de nuevos servicios de manera rápida y eficiente, respondiendo a las demandas dinámicas del mercado con mayor agilidad. Esta capacidad para adaptarse y escalar sin incurrir en gastos significativos es crucial en un entorno tecnológico en constante evolución.

1.2 Objetivos

El objetivo principal de este trabajo es explorar y entender el funcionamiento de una red definida por software (SDN) utilizando herramientas de código abierto. Específicamente, se busca:

- **Instalar y Configurar Open vSwitch:** El presente trabajo tiene como objetivo estudiar el funcionamiento de una red definida por software utilizando un switch virtual de código abierto, Open vSwitch. Este switch virtual se instalará en una máquina virtual con un sistema operativo basado en Linux, proporcionando un entorno controlado para la implementación y prueba de la arquitectura SDN. Open vSwitch es una herramienta poderosa que permite la creación y gestión de switches virtuales, soportando protocolos como OpenFlow, que es esencial para la comunicación entre el plano de control y el plano de datos en SDN.
- **Implementar y Utilizar el Controlador SDN Ryu:** Para gestionar la red, se utilizará el controlador SDN Ryu, una plataforma de control de red que facilita la gestión y orquestación de los flujos de datos a través de diferentes componentes de software y APIs. Ryu proporciona una interfaz programable que permite a los administradores definir políticas de red y gestionar el comportamiento de los dispositivos de red de manera centralizada y coherente.
- **Simulación Práctica con Raspberry Pi:** La simulación de la red se llevará a cabo conectando varias Raspberry Pi al switch virtual. Estas Raspberry Pi actuarán como dispositivos finales en la red, permitiendo la simulación de diversas situaciones y escenarios de red. Esta configuración práctica permitirá evaluar el rendimiento, flexibilidad y escalabilidad de la red definida por software en condiciones controladas y realistas.

- **Evaluación y Análisis de Resultados:** Mediante la realización de estas simulaciones, pretendo demostrar cómo la arquitectura SDN, en combinación con herramientas como Open vSwitch y el controlador Ryu, puede transformar la gestión y operación de redes, proporcionando soluciones más eficientes y adaptables. Se evaluarán diversos escenarios de red, analizando los resultados obtenidos para identificar las ventajas y posibles limitaciones de la implementación de SDN en comparación con las redes tradicionales.

En resumen, a través de este trabajo, no solo busco proporcionar una comprensión teórica de las Redes Definidas por Software, sino también ofrecer una evaluación práctica de su implementación y funcionamiento. Mi objetivo es resaltar el potencial de SDN para abordar los desafíos actuales de las redes, mejorando su flexibilidad, escalabilidad y eficiencia operativa.

1.3 Contenidos y Organización:

En este apartado, se presenta la estructura del trabajo, detallando los temas abordados y la organización de los contenidos. Se inicia con un análisis del estado del arte actual en el ámbito de las redes tradicionales, seguido de un estudio comparativo entre estas y las redes definidas por software (SDN). Además, se explorará la evolución y las perspectivas futuras de las tecnologías de red.

A continuación, se realizará una explicación detallada del entorno utilizado para la simulación del funcionamiento de una red SDN. Se abordarán conceptos clave como Open vSwitch, OpenFlow, el controlador Ryu y la integración de dispositivos como Raspberry Pi en la red.

Posteriormente, se describirán distintas situaciones prácticas diseñadas para evaluar el funcionamiento de la red SDN simulada. Se detallará el proceso de obtención y análisis de resultados, así como las conclusiones obtenidas sobre el rendimiento y la eficacia del entorno SDN.

A continuación, se presenta la organización detallada de los contenidos:

- Estado del arte
- Breve estudio y análisis de las redes tradicionales
- Aplicaciones y necesidades actuales de las redes
- Estudio de las redes definidas por software (SDN)
 - Componentes de las SDN
 - Arquitectura de las SDN
 - Ventajas y desventajas de las SDN
- Explicación teórica del entorno para la simulación del funcionamiento de una red SDN
 - Open vSwitch
 - OpenFlow
 - Controlador Ryu
 - Raspberry Pi
- Comparativa de las redes tradicionales con las redes definidas por software
- Evolución y perspectivas futuras
- Diseño de distintas situaciones prácticas para la evaluación del funcionamiento
- Obtención y análisis de resultados
- Conclusiones sobre el funcionamiento de una red SDN



Capítulo 2. Metodología del proyecto

En este apartado se presenta la metodología seguida para el desarrollo del proyecto, cuyo objetivo principal es el estudio y análisis de las Redes Definidas por Software (SDN) y su implementación práctica. El trabajo se ha estructurado en varios bloques para optimizar el tiempo y facilitar la comprensión del contenido.

El primer bloque, denominado "**Introducción y Metodología**", establece las bases del proyecto. Se incluye un capítulo introductorio donde se exponen la motivación y los objetivos que impulsan la realización del estudio, además se detalla la organización del contenido del documento. También, se describe la metodología utilizada, explicando el enfoque metodológico que se seguirá a lo largo del proyecto.

El segundo bloque, "**Marco Teórico y Estudio de Redes**", se enfoca en el estado del arte de las redes tradicionales y las Redes Definidas por Software (SDN). Se inicia con una introducción a las redes tradicionales, analizando sus fundamentos, problemas actuales, aplicaciones y perspectivas futuras. Posteriormente, se estudian las SDN, detallando su arquitectura, componentes, protocolos y estándares. También se describe el entorno de red del proyecto y se comparan las ventajas y desventajas de las SDN frente a las redes tradicionales. Finalmente, se abordan las perspectivas futuras de las SDN.

El tercer bloque, "**Estudio Práctico y Resultados**", se dedica a la implementación y evaluación práctica de una red SDN, donde se comienza con la preparación del entorno práctico, detallando los pasos para instalar y configurar los componentes necesarios. Luego, se diseñan y analizan distintos casos prácticos para evaluar la funcionalidad de la red SDN. Estas pruebas incluyen la visibilidad de la red, el control de tráfico y calidad de servicio (QoS), y la configuración de VLANs.

Esta estructura permite una comprensión clara y detallada del tema, facilitando el seguimiento del desarrollo del proyecto desde los fundamentos teóricos hasta la implementación práctica y las conclusiones finales.

Bloque de Proyecto	Actividades a Desarrollar	Inicio	Fin	Duración (días)	Duración (horas)
Bloque 1: Introducción y Metodología		20-may	23-may	4 días	20h
Capítulo 1: Introducción					
- Motivación	Identificar y redactar las razones detrás del proyecto.	20-may	20-may	1 día	
- Objetivos	Definir los objetivos específicos y generales del proyecto.	21-may	21-may	1 día	
- Contenidos y Organización	Estructurar el contenido y organización del documento.	22-may	22-may	1 día	
Capítulo 2: Metodología del Proyecto					
- Descripción de la metodología	Detallar el enfoque metodológico utilizado para el desarrollo del proyecto.	23-may	23-may	1 día	
Bloque 2: Marco Teórico y Estudio de Redes		24-may	29-may	6 días	36h
Capítulo 3: Estado del Arte de las Redes Tradicionales					
Introducción a las redes tradicionales - Problemática de las redes tradicionales - Aplicaciones y necesidades actuales - Perspectivas futuras de las redes tradicionales	Investigar y redactar sobre los fundamentos de las redes tradicionales. Identificar y explicar los problemas asociados con las redes tradicionales. Analizar las aplicaciones y necesidades modernas de las redes. Evaluar y redactar sobre las futuras tendencias y desarrollos esperados.	24-may	26-may	3 días	
Capítulo 4: Estudio de las Redes Definidas por Software (SDN)					
Fundamentos y arquitectura de las redes SDN - Entorno de red del proyecto - Ventajas y desventajas del uso de SDN frente a las redes tradicionales - Comparación de ventajas y desventajas - Perspectivas futuras de las redes SDN	Investigar y explicar la arquitectura y principios de las redes SDN. Instalar, configurar y documentar el uso del controlador Ryu, Open vSwitch y los dispositivos de red del proyecto. Comparar los pros y contras de las redes SDN con las tradicionales. Evaluar y documentar las futuras tendencias y posibles desarrollos en SDN.	26-may	29-may	3 días	
Bloque 3: Estudio Práctico y Resultados		29-may	01-jul	34 días	228h
Capítulo 5: Estudio y Resultados Prácticos de una Red SDN					
- Preparación del entorno práctico	Detallar los pasos para instalar y configurar Open vSwitch, documentar el proceso de instalación y configuración del controlador Ryu, describir la configuración necesaria de las Raspberry Pi en el entorno de red, y documentar la creación y configuración del puente virtual br0.	29-may	31-may	3 días	
- Caso 1: Conexión entre los dispositivos – Visibilidad de la red	Realizar y analizar pruebas de visibilidad de la red.	01-jun	06-jun	6 días	
- Caso 2: Configuración y Pruebas de Control de Tráfico y Calidad de Servicio (QoS)	Configurar y evaluar QoS en la red SDN.	07-jun	11-jun	5 días	
- Caso 3: Configuración y uso de VLANs	Configurar y analizar el uso de VLANs en la red SDN.	12-jun	01-jul	20 días	
Tiempo Total del Proyecto	Estudio de una red SDN mediante el switch virtual Open vSwitch y el controlador Ryu	20-may	01-jul	44 días	284h

Figura 1. Metodología del proyecto

Capítulo 3. Estado del arte de las redes tradicionales

3.1 Introducción a las redes tradicionales

El origen de las redes informáticas tradicionales se remonta a la segunda mitad del siglo XX, en concreto en el año 1960 donde surgió la idea de unir simples sistemas para que los dispositivos locales pudieran comunicarse entre sí, esto dio lugar a la creación de ARPANET, la primera red que realmente funcionó, conectando universidades y laboratorios de investigación en Estados Unidos. Con el tiempo, estas redes se volvieron más sofisticadas y adoptaron tecnologías como Ethernet y TCP/IP, que eventualmente se convirtieron en la columna vertebral de lo que hoy conocemos como Internet [4].

Al principio, las redes eran mayormente cableadas y solo podían transmitir datos de forma local, además la conexión entre estas redes era lenta y poco fiable. Pero en 1989 la invención de *World Wide Web* revolucionó la industria de internet y este se transformó en una plataforma global para compartir información, revolucionando la comunicación y el acceso a todo tipo de información [5].

En la actualidad, las redes han avanzado tanto desde sus inicios que la alta demanda y la creciente necesidad de conectividad han impulsado una evolución constante. Desde el surgimiento de internet, la sociedad ha manifestado una necesidad creciente de estar conectada, lo que ha resultado en un incremento exponencial en la cantidad de dispositivos que requieren acceso a la red. Este escenario ha generado retos significativos, especialmente en cuanto a la escalabilidad y flexibilidad de las infraestructuras de redes tradicionales. Aunque las redes tradicionales fueron efectivas en su momento, están cada vez más sobrecargadas y presentan limitaciones para adaptarse a los cambios rápidos en la demanda y a la proliferación de dispositivos conectados.

3.2 Problemática de las redes tradicionales

Una de las principales dificultades de las redes tradicionales radica en la complejidad de los protocolos de enrutamiento y conmutación que forman la base de estas redes.

Por un lado, los protocolos de enrutamiento como OSPF (*Open Shortest Path First*) [6] y BGP (*Border Gateway Protocol*) [7] requieren una alta capacidad de gestión y son propensos a configuraciones complicadas y errores humanos. OSPF, aunque es eficiente para redes internas, se vuelve complejo en entornos grandes y dinámicos debido a la necesidad de recalcular rutas constantemente utilizando el algoritmo Dijkstra [8]. BGP, por otro lado, es esencial para la operatividad de Internet, pero su configuración y mantenimiento en sistemas autónomos grandes requiere un conocimiento elevado en este ámbito y pueden ser fuentes de inestabilidad si no se gestionan adecuadamente [9].

Por otro lado, los protocolos de conmutación como STP (*Spanning Tree Protocol*) [10] y RSTP (*Rapid Spanning Tree Protocol*) [11] son necesarios para evitar bucles en la red, pero introducen complejidades adicionales ya que pueden ser lentos en converger, especialmente en redes extensas lo que afecta directamente a la disponibilidad de la red [12]. En el caso de VTP (*VLAN Trunking Protocol*) [13] y LACP (*Link Aggregation Control Protocol*) [14] facilitan la gestión y el rendimiento de la red, pero su implementación en redes tradicionales puede llegar a ser inflexible y puede llegar a requerir configuraciones manuales que aumentan la posibilidad de errores.

Cabe añadir que las decisiones tomadas tanto por los routers como por los switches al conmutar o enviar paquetes se basan en la información local disponible en el dispositivo en el momento de la decisión. Esto implica que estos dispositivos no cuentan con conocimiento del estado global de la red, como la congestión en otros puntos, rutas más óptimas disponibles o posibles fallos en otros segmentos. Por ejemplo, un router puede seleccionar una ruta según su tabla de enrutamiento actual, sin saber que existe un cuello de botella o un nodo caído más adelante en esa ruta.

Este enfoque puede resultar problemático debido a que la falta de una visión global puede llevar a decisiones subóptimas que afectan el rendimiento y la eficiencia de la red. Por ejemplo, en una red congestionada, los paquetes pueden tomar rutas menos eficientes, causando retrasos adicionales y posibles pérdidas de datos, además la falta de coordinación puede generar situaciones en las que múltiples rutas compiten por los mismos recursos, incrementando aún más la congestión y degradando la calidad del servicio.

3.3 Aplicaciones y necesidades actuales

A continuación, se presenta una muestra de las aplicaciones y necesidades actuales que las redes tradicionales deben enfrentar debido a la alta demanda y a la rápida evolución tecnológica, junto con las desventajas mencionadas en los apartados anteriores:

- **Aplicaciones en Tiempo Real:** Servicios como videoconferencias, juegos en línea y transmisión en vivo requieren una latencia mínima y alta fiabilidad. Para garantizar una experiencia de usuario óptima, estas aplicaciones demandan redes que prioricen su tráfico específico [15].
- **IoT y Dispositivos Conectados:** La creciente cantidad de dispositivos IoT exige redes capaces de manejar numerosas conexiones simultáneas y adaptarse dinámicamente a diversos tipos de tráfico. Esto incluye la compatibilidad con múltiples protocolos y estándares de comunicación [16].
- **Ambientes Empresariales y Cloud Computing:** Las redes empresariales y los entornos de computación en la nube requieren mayor seguridad, flexibilidad y escalabilidad. Es crucial que las redes empresariales soporten grandes volúmenes de datos, ofrezcan alta disponibilidad y faciliten una comunicación eficiente entre múltiples ubicaciones [17].
- **Redes Móviles y 5G:** Con la llegada del 5G, anticipa un aumento considerable en la velocidad y capacidad de las redes móviles. Aplicaciones avanzadas como la realidad aumentada (AR) y la realidad virtual (VR) necesitan redes con baja latencia y alta capacidad de datos [18].
- **Big Data:** El procesamiento y análisis de grandes volúmenes de datos requieren redes que manejen estos datos de manera rápida y eficiente. Es esencial que las redes proporcionen el ancho de banda necesario para transferencias rápidas y seguras [19].

En el contexto actual, las redes tradicionales deben transformarse para satisfacer las exigencias de las aplicaciones modernas. Esta evolución es esencial para proporcionar la flexibilidad, escalabilidad y eficiencia necesarias. Aunque las redes tradicionales han sido la base del desarrollo de la conectividad, las demandas actuales impulsan la necesidad de adoptar nuevas tecnologías. Estas deben gestionar los recursos de la red de manera más eficiente y adaptable, soportar el crecimiento exponencial del tráfico de datos y la proliferación de dispositivos conectados, garantizando así una conectividad eficiente y sostenible para el futuro.



3.4 Perspectivas futuras de las redes tradicionales

Con todo lo anterior, aunque la evolución tecnológica ha permitido grandes avances en el ámbito de las redes, la constante y creciente demanda de conectividad [20] subraya la necesidad de adoptar enfoques innovadores y adaptativos. Solo mediante la implementación de nuevas tecnologías y la modernización de las infraestructuras actuales se podrá garantizar una conectividad eficiente y sostenible en el futuro.

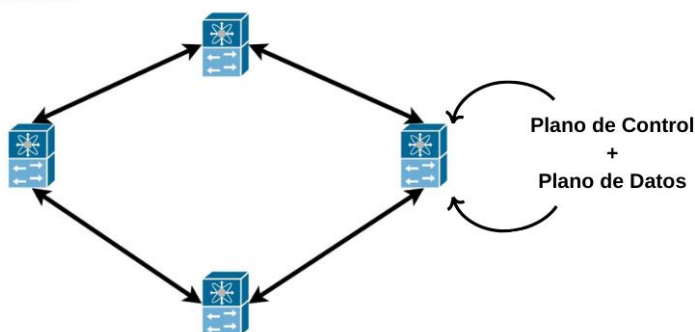
La transición hacia tecnologías más avanzadas, como las redes definidas por software (SDN), es esencial para abordar estos desafíos ya que las redes SDN permiten separar el control de la red de su hardware subyacente, proporcionando mayor agilidad y capacidad de adaptación a las necesidades cambiantes del tráfico de datos, además al centralizar el control de la red y permitir una programación directa de sus funciones, se facilita la implementación de políticas de red más flexibles y dinámicas, optimizando el uso de los recursos. Con la adopción de estas tecnologías, las redes pueden evolucionar continuamente y sin interrupciones significativas, lo que es fundamental para soportar el crecimiento exponencial del tráfico de datos y la proliferación de dispositivos conectados.

Capítulo 4. Estudio de las Redes Definidas por Software

4.1 Fundamentos y arquitectura de las redes SDN

Las redes definidas por software, más conocidas como SDN (por sus siglas en inglés, *Software-Defined Networking*), representan una revolución en el campo de las redes informáticas ya que la principal diferencia respecto a las redes tradicionales radica en que el plano de control y el plano de datos de las redes tradicionales están integrados en los mismos dispositivos (como switches y routers), sin embargo en las SDN separan estos dos planos permitiendo una gestión de la red más flexible y eficiente [21].

Redes Tradicionales



SDN

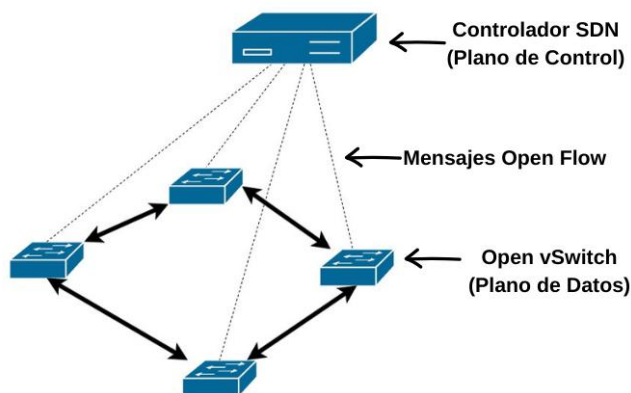


Figura 2. Plano de Control y Plano de Datos de las redes

- **Plano de control:** El plano de control es responsable de tomar decisiones sobre cómo manejar el tráfico de la red. En el caso de una SDN, el plano de control se centraliza en un controlador SDN. Este controlador tiene una visión global de la red, lo que le permite tomar decisiones más informadas y aplicar políticas de manera uniforme a través de la infraestructura de red [22].
- **Plano de datos:** El plano de datos, por otro lado, es responsable de mover el tráfico de red desde su origen hasta su destino. En una SDN, el plano de datos permanece en los dispositivos de red (como switches y routers), pero estos dispositivos reciben instrucciones del controlador SDN sobre cómo manejar el tráfico. Esto permite que los dispositivos de red sean más simples y que la lógica compleja de toma de decisiones se maneje en un solo lugar (el controlador SDN), en lugar de estar distribuida a través de toda la red [22].

Esta separación permite una mayor flexibilidad, ya que las políticas y reglas de la red pueden cambiarse rápidamente y aplicarse de manera uniforme sin necesidad de reconfigurar manualmente cada dispositivo de manera individual. La centralización del control, permite implementar de manera más efectiva optimizaciones y automatizaciones en la red, lo que mejora la eficiencia operativa y reduce el margen de error humano.

Como bien se ha mencionado, las SDN se definen como un enfoque de red donde el control está desacoplado del hardware de red, y se delega a un software llamado controlador SDN. Este controlador centraliza la gestión de la red, ofreciendo una visión global de toda la infraestructura y permite programar de forma dinámica el comportamiento de los dispositivos de red, para esto las interfaces API (Interfaces de Programación de Aplicaciones) son fundamentales en este contexto [22].

En las SDN, se utilizan principalmente dos tipos de API:

- **API Norte (Northbound):** Estas API permiten que las aplicaciones de red interactúen con el controlador SDN, facilitando la programación de la red y la gestión de políticas, ofreciendo a los administradores la capacidad de desarrollar aplicaciones que automaticen y optimicen diversas funciones de red. Por ejemplo, una API norte puede permitir a una aplicación de monitoreo obtener información en tiempo real sobre el tráfico de la red y ajustar dinámicamente las rutas para evitar congestiones [21], [23].
- **API Sur (Southbound):** Estas API permiten al controlador SDN comunicarse con los dispositivos de red, como switches y routers, en esta comunicación el controlador puede enviar instrucciones a los dispositivos para gestionar el flujo de datos de manera eficiente. Un ejemplo común de una API sur es OpenFlow, que define el protocolo y la interfaz para que los controladores SDN dicten cómo los switches deben manejar el tráfico de red [21], [23].

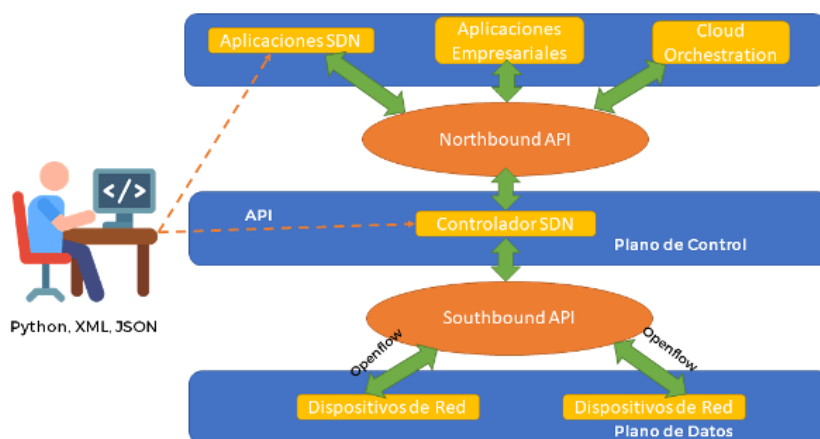


Figura 3. Interfaces API [23]

Esta arquitectura de las SDN comenzó como una respuesta a la rigidez de las redes tradicionales como bien se ha mencionado en el apartado **Capítulo 3. Estado del arte de las redes tradicionales**. Con el crecimiento exponencial de la demanda de datos y la aparición de nuevos servicios y aplicaciones, las redes convencionales mostraron limitaciones en términos de escalabilidad, flexibilidad y gestión, por ello las SDN emergieron con el fin de superar estas barreras, proporcionando una manera más ágil y adaptable de gestionar redes complejas.

4.2 Componentes de las redes SDN

4.2.1 Controlador SDN

El controlador SDN es el cerebro de una red definida por software ya que este elemento centraliza el control y se ocupa de la gestión de las políticas de red, así como de la dirección del tráfico de datos. El controlador simplifica la complejidad del hardware subyacente y proporciona una visión unificada de la red en su totalidad, lo que facilita a los administradores la programación y automatización de tareas de red [21].

Entre los controladores SDN más reconocidos se encuentran:

- **Ryu:** Es un controlador SDN de código abierto diseñado para proporcionar un marco de desarrollo sencillo y flexible para aplicaciones SDN, además es compatible con una amplia variedad de protocolos de red y se destaca por su facilidad de uso y extensibilidad, lo que lo hace ideal para desarrolladores e investigadores que trabajan en entornos de red definidos por software [24].
- **NOX/POX:** NOX es uno de los primeros controladores SDN, apreciado por su sencillez y buen rendimiento. POX, una versión derivada de NOX y escrita en Python, ofrece un entorno más amigable para quienes desarrollan aplicaciones SDN y para la investigación en este campo [25].
- **OpenDaylight:** Es una plataforma de controlador SDN de código abierto respaldada por una comunidad activa. Soporta una amplia variedad de protocolos y estándares, y es muy modular y extensible permitiendo a las organizaciones personalizar y expandir sus capacidades de red según sus necesidades específicas [26].
- **Open Network Operating System (ONOS):** Este controlador SDN de código abierto se centra en la escalabilidad y el rendimiento, por lo que se recomienda su uso para redes de proveedores de servicios y grandes infraestructuras ya que ofrece características avanzadas de resiliencia y alta disponibilidad [27].
- **Tungsten Fabric:** Tungsten Fabric se integra estrechamente con la infraestructura en la nube y es reconocido por su enfoque en la seguridad y la gestión de servicios de red complejos [28].

4.2.1.1 Interfaces API

Como se menciona en la sección **4.1 Fundamentos y arquitectura de las redes SDN**, las interfaces API (Interfaces de Programación de Aplicaciones) son fundamentales en el contexto de las SDN donde se utilizan principalmente dos tipos de API: las **API Norte** (*Northbound*) y las **API Sur** (*Southbound*). Como se puede observar en la **Figura 3. Interfaces API** [23] las API Norte permiten la interacción entre las aplicaciones de red y el controlador SDN, facilitando la programación y gestión de políticas y las API Sur permiten la comunicación entre el controlador SDN y los dispositivos de red, gestionando el flujo de datos de manera eficiente [21], [23].

4.2.2 *Switch SDN*

Los switches SDN son dispositivos clave en una red definida por software, encargados de implementar el plano de datos. Como bien se menciona en el apartado **4.2.1 Controlador SDN**, siguen las instrucciones enviadas por el controlador SDN, reenviando paquetes según las reglas de flujo establecidas. Básicamente, son los encargados de aplicar a la red las políticas y directrices definidas por el controlador.

Algunos ejemplos de switches SDN incluyen:

- **Open vSwitch (OVS):** Este es un switch virtual muy popular que soporta múltiples protocolos y es compatible con una variedad de entornos de virtualización, por ello gracias a su flexibilidad y compatibilidad, OVS es una opción preferida en entornos de virtualización, permitiendo una gestión eficaz de redes virtuales [29].
- **Cisco Nexus 9000 Series:** Esta línea de switches de Cisco soporta SDN a través de su integración con el controlador Cisco ACI (*Application Centric Infrastructure*), además son conocidos por su alto rendimiento y capacidad de escalabilidad, siendo ideales para centros de datos grandes y complejos [30].

4.2.3 *Protocolos y estándares*

Los protocolos y estándares son cruciales en el ecosistema SDN, ya que permiten la comunicación eficiente entre el controlador y los dispositivos de red, garantizando que todos los componentes trabajen de manera cohesiva.

Algunos de los principales protocolos utilizados en SDN:

- **OpenFlow:** Es el protocolo más comúnmente asociado con SDN, este protocolo define cómo el controlador comunica las reglas de enrutamiento y reenvío a los switches (virtuales o físicos), además OpenFlow facilita una separación clara entre el plano de control y el plano de datos, permitiendo una gestión centralizada y flexible de la red [31].
- **NetConf:** Es un protocolo de red utilizado para instalar, manipular y eliminar la configuración de los dispositivos de red. NetConf permite una gestión detallada y automatizada de la configuración, mejorando la eficiencia operativa [32].
- **OVSDB (*Open vSwitch Database Management Protocol*):** OVSDB permite gestionar la configuración de Open vSwitch, facilitando la integración y operación en entornos virtualizados [29].

4.3 Entorno de red del proyecto

El entorno de red planteado para el desarrollo del proyecto se ha diseñado para implementar y evaluar una red definida por software (SDN) utilizando un conjunto específico de herramientas y dispositivos. Este entorno incluye el controlador Ryu, el switch virtual Open vSwitch, y dos Raspberry Pi, donde cada componente tiene un papel crucial en la configuración y análisis de la red, proporcionando un marco robusto para estudiar las capacidades y ventajas de las redes SDN.

4.3.1 Controlador Ryu

El controlador Ryu es una plataforma de código abierto para SDN desarrollada en Python por la compañía japonesa de telecomunicaciones NTT y que a día de hoy es mantenida por la comunidad activa de desarrolladores [33].

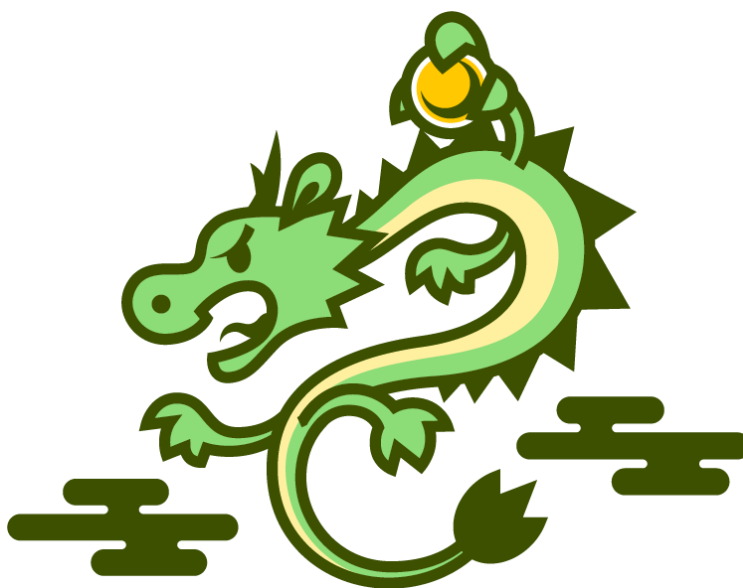


Figura 4. Controlador RYU [34]

Su principal función es actuar como el cerebro de la red SDN, tomando decisiones inteligentes sobre cómo se debe gestionar el tráfico de datos. Ryu proporciona una interfaz programable que permite a los desarrolladores crear aplicaciones de red personalizadas que pueden interactuar con los switches y routers en tiempo real, además su arquitectura modular permite a los usuarios añadir o modificar módulos según sus necesidades específicas, lo que facilita la creación de aplicaciones de red sofisticadas.

Ryu se comunica con los dispositivos de red utilizando el protocolo OpenFlow, que le permite instalar, modificar y eliminar reglas de flujo en los switches. Estas reglas determinan cómo deben manejarse los paquetes de datos que atraviesan la red. Además, Ryu puede recopilar estadísticas de los dispositivos de red, lo que le permite tomar decisiones basadas en datos en tiempo real para optimizar el rendimiento de la red [34].

Como se nos indica en su manual de instalación y configuración [35], Ryu puede detectar congestión en un enlace de red y redirigir el tráfico a través de una ruta alternativa menos congestionada, además también puede implementar políticas de seguridad para bloquear tráfico no autorizado o sospechoso y garantizar así la calidad del servicio (QoS) para aplicaciones críticas.

Durante el desarrollo del proyecto el controlador Ryu tiene compatibilidad con los siguientes switches:

	OpenFlow 1.3	OpenFlow 1.4
Allied Telesis x510	Supported	
Allied Telesis x930	Supported	
Centec V350	Supported	
CpQd	Supported	
Edge-Core AS4600-54T	Supported	
HP 2920	Supported	
IBM RackSwitch G8264	Supported	
Indigo Virtual Switch	Supported	
Lagopus	Supported	
LINC	Supported	Supported
NEC PF5220	Supported	
NoviFlow NoviKit200	Supported	
Open vSwitch	Supported	Supported
Open vSwitch (netdev)	Supported	Supported
Pica8 P-3290	Supported	Supported
Trema Switch	Supported	

Figura 5. Compatibilidad de RYU con switches [34]

4.3.2 Switch virtual – Open vSwitch

El switch utilizado para implementar y evaluar una red SDN es el switch virtual Open vSwitch (OVS) [29]. Fue desarrollado por Nicira (ahora parte de la empresa VMWare [36]) con el propósito de ser un switch virtual de código abierto con una gran integración con software de virtualización. Open vSwitch juega un papel vital en las redes SDN al proporcionar una plataforma flexible y escalable para el manejo del tráfico de red. OVS es altamente configurable y permite la implementación de reglas de flujo detalladas que especifican cómo se deben procesar los paquetes de datos [29].

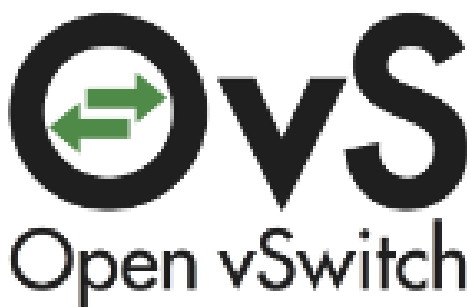


Figura 6. Open vSwitch logo [29]

OVS opera en el nivel de software, proporcionando capacidades de conmutación avanzadas que normalmente se encuentran en hardware especializado. Esto incluye soporte para VLANs, etiquetado de QoS, y balanceo de carga, además puede ser controlado y configurado dinámicamente por un controlador SDN utilizando el protocolo OpenFlow [29].

4.3.2.1 Reglas de flujo

Como se especifica en la documentación de oficial de Open vSwitch [29], las reglas de flujo en Open vSwitch (OVS) son esenciales para el funcionamiento óptimo de las redes definidas por software (SDN). Estas reglas actúan como instrucciones específicas que determinan cómo deben manejarse los paquetes de datos que atraviesan el switch, permitiendo un control detallado y flexible del tráfico de red.

Con estas reglas, el switch puede realizar tareas avanzadas como el reenvío de datos al destino correcto, el filtrado del tráfico no deseado, la modificación de la información en los paquetes, el balanceo de carga entre diferentes rutas, la garantía de calidad del servicio (QoS), así como la monitorización y encapsulación de datos para mantener todo bajo control.

Las reglas de flujo se configuran a través del controlador SDN el cual utiliza el protocolo OpenFlow para comunicarse con el switch. Cada regla de flujo en OVS puede especificar una serie de coincidencias (*matches*) y acciones (*actions*):

- **Matches:** Son los criterios que determinan qué paquetes coinciden con la regla y estos pueden incluir:
 - Direcciones IP de origen y destino.
 - Puertos de origen y destino.
 - Tipo de protocolo (TCP, UDP, etc.).
 - VLAN ID.
 - Direcciones MAC de origen y destino.
- **Actions:** Son las acciones a realizar sobre los paquetes que coinciden con los criterios de la regla y pueden ser:
 - Reenviar el paquete a un puerto específico.
 - Modificar los campos del encabezado del paquete (por ejemplo, cambiar la dirección IP de destino).
 - Enviar el paquete al controlador SDN.
 - Descartar el paquete.

4.3.3 OpenFlow

El protocolo utilizado para el desarrollo del proyecto es OpenFlow, el cual fue desarrollado inicialmente por la Universidad de Stanford [37]. Este protocolo de comunicación estándar es utilizado entre el controlador SDN y los switches de red, como Open vSwitch. OpenFlow fue uno de los primeros protocolos desarrollados específicamente para SDN y ha sido fundamental en la adopción y evolución de esta tecnología ya que permite a los controladores SDN comunicarse directamente con los planos de datos de los dispositivos de red, permitiendo a los administradores de red programar el comportamiento de los switches y routers de manera dinámica. OpenFlow permite un control directo y granular sobre los dispositivos de red, facilitando la implementación de SDN [31].



Figura 7. Open vSwitch logo [31]

El protocolo OpenFlow define un conjunto de mensajes que permiten al controlador SDN y a los switches intercambiar información sobre el estado de la red y las reglas de flujo. Estos mensajes incluyen comandos para agregar, modificar o eliminar reglas de flujo, así como para recopilar estadísticas y supervisar el rendimiento del switch [31].

4.3.4 Dispositivos de red – Raspberry Pi

Las Raspberry Pi fueron desarrolladas por la Fundación Raspberry Pi con el objetivo de promover la enseñanza de ciencias de la computación básicas en escuelas y países en desarrollo. Desde su lanzamiento, han ganado popularidad en una amplia gama de aplicaciones debido a su versatilidad, el bajo coste y el soporte de la comunidad de desarrolladores [38].

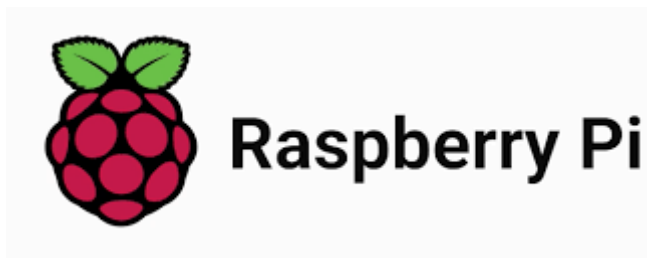


Figura 8. Raspberry Pi logo [38]

En este proyecto, se ha seleccionado estas dos Raspberry Pi por su bajo coste, versatilidad y rendimiento, además se utilizan como nodos finales en la red SDN, con el fin de generar tráfico de datos y simular diversos escenarios de red, permitiendo una evaluación exhaustiva de las reglas de flujo y la capacidad de gestión de Ryu y Open vSwitch.

4.4 Ventajas y desventajas del uso de SDN frente a las redes tradicionales

Teniendo en cuenta la problemática, las aplicaciones y necesidades actuales, así como las perspectivas futuras de las redes tradicionales desarrollado en el **Capítulo 3 Estado del arte de las redes tradicionales**, y con las ventajas y desventajas encontradas durante el estudio de las SDN en el **Capítulo 4 Estudio de las Redes Definidas por Software** se puede hacer una comparativa de los puntos más importantes:

4.4.1 *Ventajas y Desventajas de las SDN*

Las redes definidas por software (SDN) ofrecen ventajas que las hacen muy atractivas para muchas organizaciones, por ejemplo, una de las principales es el control centralizado, que permite una gestión de la red mucho más eficiente y simplificada, además, al tener un plano de datos basado en software, se facilita la implementación de cambios y actualizaciones sin necesidad de modificar el hardware. La configuración automatizada también es un gran beneficio, ya que reduce el tiempo y el esfuerzo necesarios para configurar la red, minimizando los errores humanos [39].

Otro aspecto importante es el bajo coste, ya que las SDN utilizan hardware genérico, esto reduce significativamente los costos de implementación y mantenimiento., además las SDN también son altamente escalables y flexibles, permitiendo realizar ajustes rápidos y dinámicos en la configuración de la red. La seguridad mejorada es otro punto a favor, ya que el control centralizado permite una respuesta más rápida y efectiva a las amenazas de seguridad. La gestión simplificada facilita la administración de la red desde un único punto de control, y el despliegue rápido permite la implementación de nuevos servicios y configuraciones de red de manera ágil y eficiente [39].

Sin embargo, las SDN también tienen sus desventajas, como, por ejemplo, la complejidad de implementación es una de las principales, ya que requiere conocimientos técnicos especializados y puede ser difícil de integrar con las infraestructuras existentes. Además, el diseño de las SDN conlleva un riesgo de punto único de fallo debido a la centralización del control, lo que puede convertirse en un punto crítico de vulnerabilidad, además debido a esta centralización también se amplía la superficie de ataque, introduciendo nuevas vulnerabilidades, especialmente en el plano de control [39].

La dependencia del software es otra desventaja, ya que la fiabilidad del sistema depende en gran medida de la estabilidad y seguridad del software del controlador por lo que lo más importante será tener el software actualizado con los últimos parches de seguridad y en la última versión disponible del mismo. [39].

4.4.2 *Ventajas y Desventajas de las Redes Tradicionales*

Las redes tradicionales también tienen sus propias ventajas, una de ellas es el control distribuido, donde cada dispositivo de red gestiona su propio control, lo que en algunos casos puede aumentar la resiliencia. Además, el plano de datos basado en hardware ofrece un rendimiento predecible y generalmente más robusto en entornos estáticos. La configuración manual de los dispositivos permite un control detallado y minucioso de cada aspecto de la configuración de la red [39].

Otra ventaja es la estabilidad, ya que las redes tradicionales suelen conocerse en profundidad y cuentan con protocolos y métodos establecidos. También es importante mencionar que estas redes tienen una alta compatibilidad con dispositivos y tecnologías existentes, lo que evita la necesidad de realizar grandes cambios [39].

Sin embargo, las redes tradicionales también presentan desventajas significativas, una de ellas es por ejemplo el coste elevado, ya que requieren hardware propietario y especializado, lo que incrementa los costos de implementación y mantenimiento [39].

Otra desventaja es la limitada escalabilidad, y expandir la red puede ser un proceso complejo y caro debido a esta dependencia del hardware [39].

Además, la flexibilidad es reducida, lo que dificulta realizar cambios dinámicos y rápidos en la red, así como también la seguridad puede ser complicada de gestionar, ya que la seguridad descentralizada hace más difícil la implementación de políticas coherentes y rápidas. Por otro lado, la gestión, puede ser bastante compleja, ya que la administración distribuida requiere una mayor coordinación y recursos. Por último, el despliegue es lento y puede hacer que la implementación de nuevos servicios y configuraciones de red sea un proceso laborioso y prolongado [39].

Aspecto	SDN	Red Tradicional
Plano de Control	Centralizado	Distribuido
Plano de Datos	Basado en software	Basado en hardware
Configuración	Automatizada	Manual
Coste	Bajo (usando hardware genérico)	Alto (requiere hardware propietario)
Escalabilidad	Alta	Limitada
Flexibilidad	Alta	Limitada
Seguridad	Mejorada (control centralizado)	Difícil de gestionar (control descentralizado)
Gestión	Simplificada (gestión centralizada)	Compleja (gestión distribuida)
Tiempo de Despliegue	Rápido	Lento

Tabla 1. Ventajas y desventajas SDN vs Redes Tradicionales [39]

4.5 Perspectivas futuras de las redes SDN

Con todo el conocimiento obtenido durante la fase de investigación de este proyecto del funcionamiento, configuración y ventajas del uso de SDN frente a las redes tradicionales, el futuro de las redes definidas por software (SDN) parece prometedor, además teniendo en cuenta las perspectivas futuras desarrolladas en el apartado **3.4 Perspectivas futuras de las redes tradicionales** así como las necesidades actuales desarrolladas en el **punto 3.3 Aplicaciones y necesidades actuales** podemos afirmar que las SDN tenderán a desarrollar avances en los siguientes sectores:

- **Integración con 5G:** Una de las tendencias más destacadas es la integración de SDN con la tecnología 5G, ya que se espera que las SDN jueguen un papel crucial en el despliegue de esta tecnología, proporcionando una conectividad mejorada y reduciendo la latencia. Esta combinación permitirá una mayor agilidad y flexibilidad en la gestión de redes, adaptándose rápidamente a las demandas cambiantes actuales [40].
- **Impacto en el Internet de las Cosas (IoT):** En un entorno donde la cantidad de dispositivos conectados sigue creciendo exponencialmente, la capacidad de SDN para manejar y optimizar estas conexiones será indispensable, por lo que las SDN también tendrá un impacto significativo en el Internet de las Cosas (IoT), permitiendo conexiones más directas y eficientes entre los dispositivos IoT y las aplicaciones en la nube. Esta capacidad mejorará la gestión y el rendimiento de las redes IoT, asegurando una mejor utilización de los recursos y una respuesta más rápida a las demandas de tráfico [41].
- **Avances en la seguridad y la automatización:** El enfoque centralizado de SDN mejora la seguridad de la red al proporcionar una visión holística y la capacidad de implementar políticas de seguridad coherentes y dinámicas. Frente a las amenazas de la seguridad las SDN permiten una respuesta rápida y efectiva, reduciendo los riesgos y mejorando la protección de la red. Además, la capacidad de automatización de las SDN permite una configuración y gestión más eficientes, minimizando la intervención manual y los errores humanos, además con esto no solo se reducen los costes, sino que también se incrementa la rapidez y la eficiencia operativa [42].
- **Aplicación en diversos sectores:** Las aplicaciones de SDN están expandiéndose en numerosos sectores. Por ejemplo, en los centros de datos, las SDN facilitan la gestión de grandes volúmenes de servidores y conmutadores, permitiendo cambios rápidos y reduciendo el tiempo de inactividad. En entornos corporativos, las SDN permiten una gestión centralizada y coherente de múltiples edificios y sucursales, asegurando que todas las partes de la organización operen bajo las mismas políticas y configuraciones. Además, en entornos cloud, las SDN permiten la asignación dinámica de recursos, mejorando la eficiencia y reduciendo costos [43].

Teniendo en cuenta los puntos anteriores, el futuro de las SDN parece prometedor, aunque ciertos desafíos como la complejidad de la implementación y la interoperabilidad entre productos de diferentes proveedores deberán ser abordados. Esto son cuestiones importantes y de primeras la transición hacia las SDN puede requerir una inversión considerable, aunque esta transformación permitirá a las organizaciones responder de manera más efectiva el mercado y mejorar significativamente sus necesidades.

Capítulo 5. Estudio y resultados prácticos de una red SDN

5.1 Preparación del entorno práctico

A continuación, vamos a proceder con la preparación del entorno práctico, en este apartado se detallará paso por paso como se ha instalado y configurado cada uno de los componentes de nuestro esquema de red tanto la parte de hardware como la parte de software.

En primer lugar, para tener una visión mucho más clara de cómo se conectarán los componentes, se ha realizado el siguiente esquema:

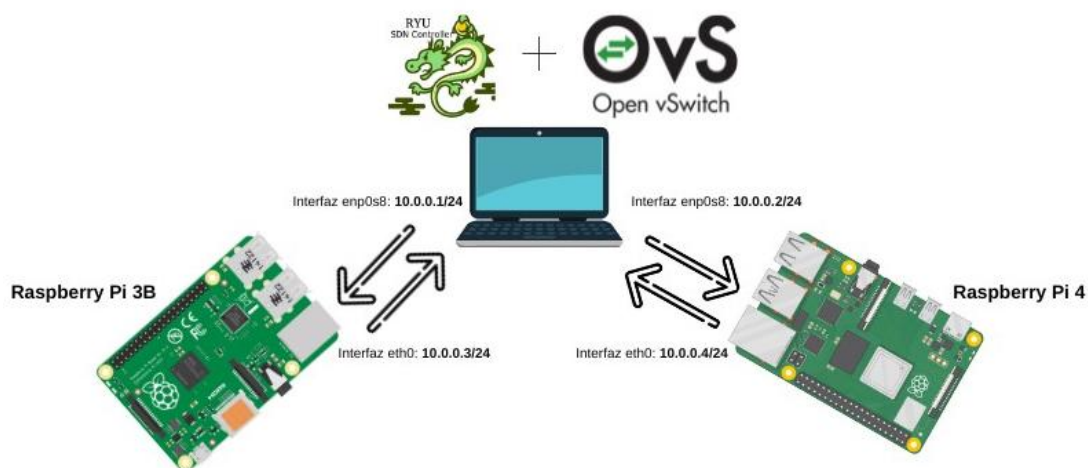


Figura 9. Esquema del proyecto

Una vez se conoce el esquema que seguirá la parte práctica del proyecto tendremos que seguir los siguientes pasos:

Primero, nos encargaremos de instalar el switch virtual Open vSwitch en una máquina Linux. Este switch será el núcleo central que gestionará el tráfico de red en nuestra configuración.

Luego, procederemos con la instalación del controlador Ryu. Este controlador se encargará de gestionar y controlar toda la red SDN, definiendo las políticas de enrutamiento y gestión del tráfico. Podemos instalar Ryu en la misma máquina Linux donde se encuentra Open vSwitch o en una máquina dedicada para ello.

El siguiente paso incluye la configuración de las dos Raspberry Pi, que utilizaremos para analizar cómo funciona la red. Cada Raspberry Pi necesita tener un sistema operativo basado en Linux y estar configurada para conectarse al switch virtual. Además, les instalaremos herramientas necesarias para realizar pruebas de red y capturar tráfico. Estas herramientas nos ayudarán a medir el rendimiento de la red y capturar datos para su posterior análisis.

Una vez que todos los componentes estén instalados y configurados, conectaremos las Raspberry Pi al switch virtual Open vSwitch mediante las interfaces ethernet físicas de las mismas.

Con todos los componentes interconectados y configurados, nuestro entorno de pruebas estará listo para empezar el análisis del funcionamiento de la red SDN. Este esquema y las configuraciones detalladas nos proporcionan una base sólida para implementar nuestra red, permitiéndonos explorar y entender su comportamiento de manera práctica y efectiva.

5.1.1 Instalación y configuración de Open vSwitch

En primer lugar, se procederá a la instalación y configuración del switch virtual Open vSwitch, para ello primero se necesitará una máquina virtual Linux. En este proyecto se utilizará la versión 24.0.4 LTS de Ubuntu [44] y se ejecutará usando el software de virtualización VirtualBox [45].

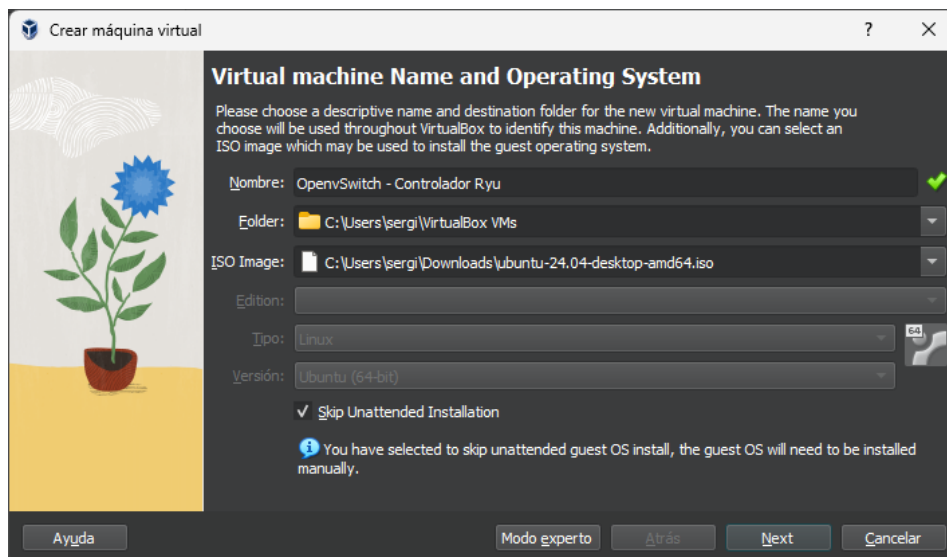


Figura 10. Creación de la máquina virtual (I)

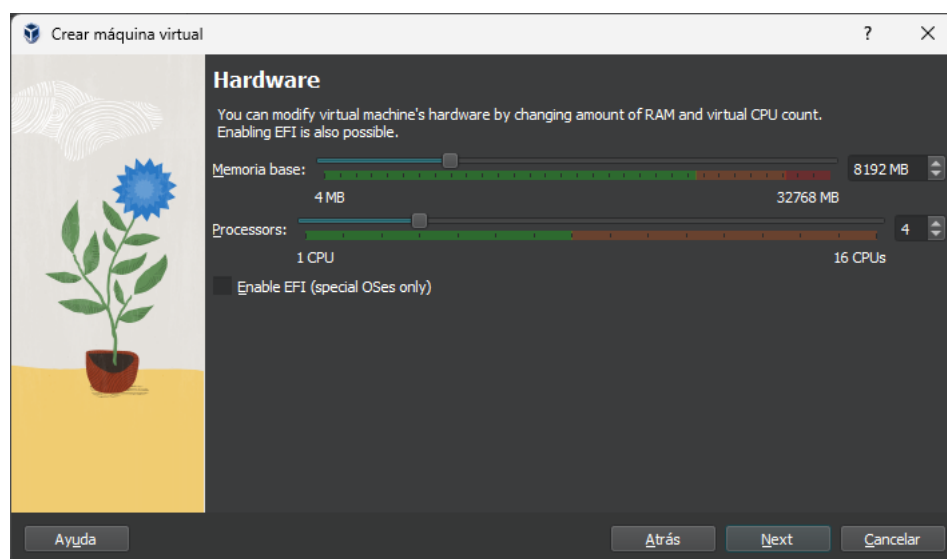


Figura 11. Creación de la máquina virtual (II)

Una vez creada la máquina virtual donde se ejecutará el switch virtual Open vSwitch, hay que configurar el sistema operativo mediante las configuraciones de personalización que nos ofrece el asistente de instalación.

Cree su cuenta

Cree su cuenta

Su nombre
openvswitch ✓

El nombre del equipo
openvswitch ✓

Elija un nombre de usuario
openvswitch ✓

Elija una contraseña
openvswitch Ocultar Contraseña débil

Confirme su contraseña
openvswitch ✓

Solicitar mi contraseña para acceder

Utilizar Active Directory

Figura 12. Asistente de instalación Ubuntu 24.0.4 LTS

Finalizado el proceso de instalación del sistema operativo, se procederá a la instalación del switch virtual Open vSwitch siguiendo la guía de instalación del sitio oficial [29]. Como se indica en la documentación oficial recomiendan tener actualizado todos los paquetes instalados en el sistema operativo con el fin de no tener problemas de compatibilidad con la versión, por ello se ejecutarán los siguientes comandos:

```
sudo apt update  
sudo apt upgrade
```

Figura 13. Comandos de actualización de software del sistema

Cuando se hayan actualizado los paquetes del sistema se procederá con la instalación de Open vSwitch mediante el siguiente comando:

```
sudo apt install openvswitch-switch
```

Figura 14. Comando de instalación de Open vSwitch

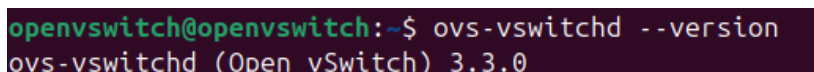
Para comprobar que el switch virtual Open vSwitch ha sido instalado correctamente se debe ejecutar el siguiente comando:



```
ovs-vswitchd --version
```

Figura 15. Comando para comprobar versión de Open vSwitch

Cuando se ejecute el comando anterior, se debería recibir la siguiente respuesta:



```
openvswitch@openvswitch:~$ ovs-vswitchd --version  
ovs-vswitchd (Open vSwitch) 3.3.0
```

Figura 16. Versión de Open vSwitch instalada

Como se puede comprobar, Open vSwitch ha sido instalado correctamente en la máquina Linux en la última versión disponible en el momento de realizar este proyecto. Con todo esto solamente restará crear el puente virtual entre el switch virtual Open vSwitch y las dos Raspberry Pi que componen nuestro esquema de red, así como la creación de las reglas de flujo del switch virtual. Este proceso se abordará en el apartado **5.1.4 Creación del puente virtual br0**.

5.1.2 Instalación y configuración del controlador Ryu

Cuando se hayan completado los pasos indicados en el apartado anterior, se procederá a la instalación del controlador de la red SDN. En este proyecto, utilizaremos el controlador SDN Ryu.

Para utilizar el controlador Ryu en el esquema de red, es necesario tener instalado el lenguaje de programación Python en el sistema donde se instalará dicho controlador, tal como indica la guía de instalación ubicada en su sitio oficial [34]. En este caso, la máquina que funcionará como switch virtual también actuará como controlador de la red SDN. Por lo tanto, se debe verificar que Python esté instalado en dicha máquina.

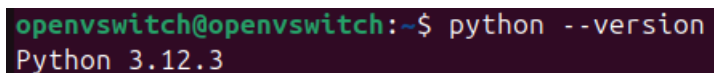
Para ello, se debe ejecutar el siguiente comando:



```
python --version
```

Figura 17. Comando para comprobar versión de Python

Una vez ejecutado el comando, la respuesta debería ser similar a la siguiente:



```
openvswitch@openvswitch:~$ python --version  
Python 3.12.3
```

Figura 18. Versión de Python por defecto del sistema

Con la respuesta recibida se comprueba que la versión de Python instalada por defecto en el sistema no es compatible con el controlador Ryu en la fecha en la que se realiza este proyecto. Según se menciona en el repositorio oficial de GitHub [46], el controlador Ryu solo es compatible con la versión 3.9 de Python.

Para resolver este problema, la solución más viable es instalar una versión específica de Python utilizando la herramienta pyenv [47], lo que evitará problemas de compatibilidad. Siguiendo la guía de instalación disponible en el repositorio de GitHub [47], se ha procedido a instalar pyenv en la máquina Linux. Tras esta instalación, se procederá a instalar la versión compatible con el controlador Ryu mediante el siguiente comando:

```
pyenv install 3.9.0
```

Figura 19. Comando instalación Pyenv - Python 3.9.0

Cuando se ejecute este comando, se debería recibir una respuesta similar a la siguiente:

```
openvswitch@openvswitch:~$ pyenv install 3.9.0
Downloading Python-3.9.0.tar.xz...
-> https://www.python.org/ftp/python/3.9.0/Python-3.9.0.tar.xz
Installing Python-3.9.0...
patching file Misc/NEWS.d/next/Build/2021-10-11-16-27-38.bpo-45405.iSfdW5.rst
patching file configure
patching file configure.ac
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/bz2.py", line 18, in <module>
    from _bz2 import BZ2Compressor, BZ2Decompressor
ModuleNotFoundError: No module named '_bz2'
WARNING: The Python bz2 extension was not compiled. Missing the bzip2 lib?
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/curses/__init__.py", line 13, in <module>
    from _curses import *
ModuleNotFoundError: No module named '_curses'
WARNING: The Python curses extension was not compiled. Missing the ncurses lib?
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'readline'
WARNING: The Python readline extension was not compiled. Missing the GNU readline lib?
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/tkinter/__init__.py", line 37, in <module>
    import _tkinter # If this fails your Python may not be configured for Tk
ModuleNotFoundError: No module named '_tkinter'
WARNING: The Python tkinter extension was not compiled and GUI subsystem has been detected. Missing the Tk toolkit?
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/lzma.py", line 27, in <module>
    from _lzma import *
ModuleNotFoundError: No module named '_lzma'
WARNING: The Python lzma extension was not compiled. Missing the lzma lib?
Installed Python-3.9.0 to /home/openvswitch/.pyenv/versions/3.9.0
```

Figura 20. Proceso de instalación Python 3.9.0

Para que la versión de Python utilizada en el sistema sea la versión 3.9.0 y no la versión que venía por defecto instalada en el sistema operativo Ubuntu 24.0.4 LTS, se debe ejecutar el siguiente comando:

```
pyenv global 3.9.0
```

Figura 21. Comando para establecer uso de la versión de Python 3.9.0 en el sistema

Establecida esta versión de Python como la versión que se ejecutará en el sistema a partir de ahora, será necesario instalar la versión de pip acorde a Python 3.9.0. Pip es el gestor de paquetes para Python y se utiliza para instalar y gestionar bibliotecas y paquetes adicionales que no forman parte de la biblioteca estándar de Python [48]. Para obtener esta versión específica de pip, se utilizará la herramienta get-pip.py [49]. Para ello, se ejecutará el siguiente comando de instalación:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
python3.9 get-pip.py
```

Figura 22. Instalación herramienta get-pip.py

Tras ejecutar este comando, se debería recibir una respuesta similar a la siguiente:

```
openvswitch@openvswitch:~$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
% Total % Received % Xferd Average Speed Time Time Time Current  
 Dload Upload Total Spent Left Speed  
100 2574k 100 2574k 0 0 2357k 0 0:00:01 0:00:01 ---:--:-- 2359k  
openvswitch@openvswitch:~$ python3.9 get-pip.py  
Collecting pip  
Downloading pip-24.0-py3-none-any.whl.metadata (3.6 kB)  
Collecting wheel  
Downloading wheel-0.43.0-py3-none-any.whl.metadata (2.2 kB)  
Downloading pip-24.0-py3-none-any.whl (2.1 MB)  
----- 2.1/2.1 MB 4.4 MB/s eta 0:00:00  
Downloading wheel-0.43.0-py3-none-any.whl (65 kB)  
----- 65.8/65.8 kB 2.1 MB/s eta 0:00:00  
Installing collected packages: wheel, pip  
Attempting uninstall: pip  
Found existing installation: pip 20.2.3  
Uninstalling pip-20.2.3:  
Successfully uninstalled pip-20.2.3  
Successfully installed pip-24.0 wheel-0.43.0
```

Figura 23. Proceso de instalación de la herramienta get-pip.py

Para comprobar que la versión de Python, así como la versión de pip son correctas y compatibles con el controlador Ryu, se pueden ejecutar los siguientes comandos:

```
openvswitch@openvswitch:~$ python3.9 --version
Python 3.9.0
openvswitch@openvswitch:~$ pip --version
pip 24.0 from /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages/pip (python 3.9)
```

Figura 24. Versiones de pip y Python compatibles con Ryu

Con todo el procedimiento anterior, el sistema operativo ya está listo para la instalación del controlador Ryu siguiendo la guía oficial [34]. Tal y como se indica en el repositorio oficial [46], se deberán ejecutar los siguientes comandos:

```
git clone https://github.com/faucetsdn/ryu.git
cd ryu; pip install .
```

Figura 25. Comandos de instalación de Ryu

```
Processing /home/openvswitch/ryu
  Preparing metadata (setup.py) ... done
Collecting eventlet==0.31.1 (from ryu==4.34)
  Downloading eventlet-0.31.1-py2.py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: msgpack>=0.4.0 in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from eventlet==0.31.1)
Requirement already satisfied: netaddr in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from eventlet==0.31.1)
Requirement already satisfied: oslo.config>=2.5.0 in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from eventlet==0.31.1)
Requirement already satisfied: ovs>=2.6.0 in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from eventlet==0.31.1)
Collecting packaging==20.9 (from ryu==4.34)
  Downloading packaging-20.9-py2.py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: routes in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from packaging==20.9)
Requirement already satisfied: six>=1.4.0 in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from packaging==20.9)
Collecting tinyrpc==1.0.4 (from ryu==4.34)
  Downloading tinyrpc-1.0.4.tar.gz (26 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: webob>=1.2 in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from tinyrpc==1.0.4)
Collecting dnspython<2.0.0,>=1.15.0 (from eventlet==0.31.1->ryu==4.34)
  Downloading dnspython-1.16.0-py2.py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: greenlet>=0.3 in /home/openvswitch/.pyenv/versions/3.9.0/lib/python3.9/site-packages (from dnspython<2.0.0,>=1.15.0)
Collecting pyparsing>=2.0.2 (from packaging==20.9->ryu==4.34)
  Downloading pyparsing-3.1.2-py3-none-any.whl.metadata (5.1 kB)
```

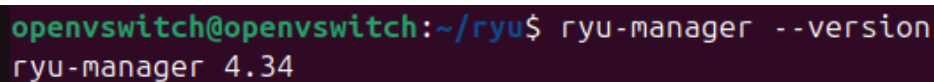
Figura 26. Proceso de instalación de Ryu

Cuando finalice el proceso de instalación del controlador de la red SDN Ryu, se podrá comprobar la versión tanto de Ryu como del switch virtual Open vSwitch mediante los siguientes comandos:



```
ryu-manager --version
```

Figura 27. Comprobar versión de Ryu instalada



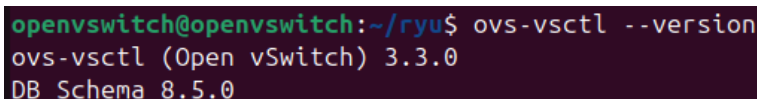
```
openvswitch@openvswitch:~/ryu$ ryu-manager --version  
ryu-manager 4.34
```

Figura 28. Versión de Ryu instalada en el sistema



```
ovs-vsctl --version
```

Figura 29. Comprobar versión de Open vSwitch instalada



```
openvswitch@openvswitch:~/ryu$ ovs-vsctl --version  
ovs-vsctl (Open vSwitch) 3.3.0  
DB Schema 8.5.0
```


Figura 30. Versión de Open vSwitch instalada en el sistema

Como se observa, los comandos que se han utilizado son útiles para verificar las versiones de software en un entorno de red SDN. El comando **ryu-manager --version** muestra qué versión del controlador Ryu se tiene instalada.

Por otro lado, el comando **ovs-vsctl --version** muestra la versión de Open vSwitch (OVS) instalada en tu sistema, junto con la versión del esquema de la base de datos que usa.

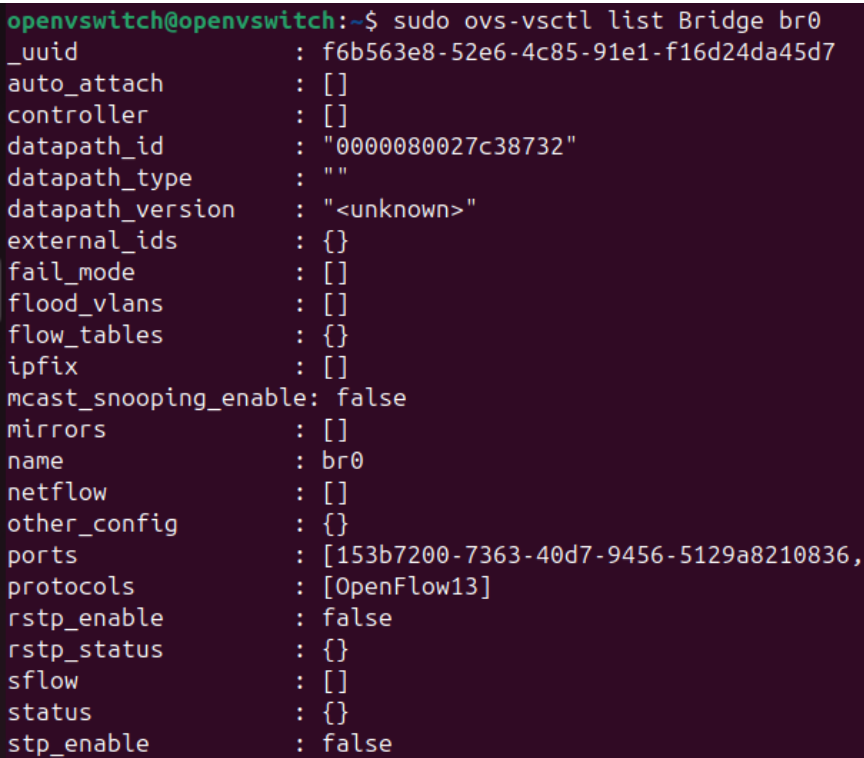
En el caso de la fecha de realización de este proyecto la versión de Ryu, Open vSwitch y DB Schema son 4.34, 3.3.0 y 8.5.0 respectivamente.

Para conseguir que funcione el controlador Ryu, hay que asignarlo como controlador en Open vSwitch. Para comprobar si hay algún controlador asignado a Open vSwitch se debe utilizar este comando:



```
sudo ovs-vsctl list Bridge br0
```

Figura 31. Obtención de información del bridge br0 de Open vSwitch



```
openvswitch@openvswitch:~$ sudo ovs-vsctl list Bridge br0
_name                : br0
_datapath_id         : "0000080027c38732"
_datapath_type        : ""
_datapath_version    : "<unknown>"
_external_ids         : {}
_fail_mode            : []
_flood_vlans          : []
_flow_tables          : {}
_ipfix                : []
_mcast_snooping_enable: false
_mirrors              : []
_netflow              : []
_other_config         : {}
_ports                : [153b7200-7363-40d7-9456-5129a8210836,
_protocols            : [OpenFlow13]
_rstp_enable          : false
_rstp_status          : {}
_sflow                : []
_status               : {}
_stp_enable           : false
```

Figura 32. Información del bridge br0

Como se observa en el campo **controller** no hay ningún controlador SDN asociado a Open vSwitch. Para enlazar el controlador Ryu instalado en el sistema, se debe tener Ryu ejecutándose, para ello se debe utilizar el siguiente comando:



```
ryu-manager simple_switch_13.py
```

Figura 33. Comando de ejecución de Ryu

En este caso se observa que Ryu se debe ejecutar pasándole como argumento un script escrito en Python. El uso de scripts en Python permite definir el comportamiento del controlador de red donde los usuarios tienen la posibilidad de escribir sus propios scripts personalizados. Además, como en este caso Ryu ofrece scripts de prueba predeterminados para facilitar el aprendizaje y las pruebas en los entornos de red SDN, y en este caso se puede obtener del repositorio oficial [46].

El script **simple_switch_13.py** define una aplicación básica de Ryu que actúa como un controlador de red SDN utilizando OpenFlow 1.3. Al iniciar, la aplicación se prepara para manejar eventos de OpenFlow, específicamente el evento **EventOFPSwitchFeatures**, que se dispara cuando un switch se conecta al controlador.

Al recibir este evento, el controlador instala una regla de flujo predeterminada en el switch que coincide con todos los paquetes y los envía al controlador. Esta regla se instala con una prioridad de 0, lo que indica que es una regla de baja prioridad y se aplicará en ausencia de otras reglas más específicas. El método **add_flow** se encarga de crear y enviar esta regla de flujo al switch, utilizando la API de OpenFlow proporcionada por Ryu.

Como resultado, todos los paquetes que lleguen al switch se reenviarán al controlador, permitiendo una inspección y manejo centralizado de todo el tráfico de red. Este comportamiento es esencial para controladores SDN que requieren visibilidad completa del tráfico antes de aplicar políticas más detalladas, para más información del script y del funcionamiento del controlador se puede consultar la página y repositorio oficiales [34] y [46].

La ejecución del comando de la **Figura 33. Comando de ejecución de Ryu** muestra un resultado como este:

```
openvswitch@openvswitch:~/Escritorio/Ryu_scripts$ ryu-manager --verbose simple_switch_13.py
loading app simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app simple_switch_13.py of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoRequest
connected socket:<eventlet.greenio.base.GreenSocket object at 0x75e678b7e610> address:('127
.0.0.1', 55648)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x75e678b7ed30>
move onto config mode
EVENT ofp_event->SimpleSwitch EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x67d59610,OFPSwitchFeatures(a
uxiliary_id=0,capabilities=79,datapath_id=8796760147762,n_buffers=0,n_tables=254)
move onto main mode
```

Figura 34. Ejecución de ryu-manager simple_switch_13.py

Tras lanzar Ryu se debe verificar que hay un proceso ejecutándose en el puerto **6633** tal y como se indica en el repositorio oficial [46]. Para ello se utiliza el siguiente comando:

```
sudo netstat -nlp | grep 6633
```

Figura 35. Comando de verificación de la ejecución de Ryu en el puerto 6633

```
openvswitch@openvswitch:~$ sudo netstat -nlp | grep 6633
tcp        0      0 0.0.0.0:6633          0.0.0.0:*            ESCUCHAR   15800/python3.9
tcp        0      0 0.0.0.0:6633          0.0.0.0:*            ESCUCHAR   11690/python3.9
```

Figura 36. Proceso ejecutándose en el puerto 6633

Tras comprobar que hay un proceso ejecutándose en el puerto **6633** se debe asignar el controlador Ryu a Open vSwitch mediante el siguiente comando:

```
sudo ovs-vsctl set-controller br0 tcp:127.0.0.1:6633
```

Figura 37. Asignación de Ryu a Open vSwitch

```
sudo ovs-vsctl get-controller br0
```

Figura 38. Comprobación del controlador asignado a Open vSwitch

```
openvswitch@openvswitch:~$ sudo ovs-vsctl get-controller br0
tcp:127.0.0.1:6633
```

Figura 39. Asignación correcta del controlador

```
openvswitch@openvswitch:~$ sudo ovs-vsctl list Bridge br0
_uuid                : f6b563e8-52e6-4c85-91e1-f16d24da45d7
auto_attach          : []
controller            : [9790b3e8-4ccf-4be6-a89a-7f3d917a7ea6]
datapath_id           : "0000080027c38732"
datapath_type         : ""
datapath_version      : "<unknown>"
external_ids          : {}
fail_mode             : []
flood_vlans           : []
flow_tables           : {}
ipfix                 : []
mcast_snooping_enable : false
mirrors               : []
name                  : br0
netflow               : []
other_config          : {}
ports                 : [153b7200-7363-40d7-9456-5129a8210836, 28b61757-ee62-44ac-a620-9e99e6
de3f6a, 53e4f340-09cf-4de1-97a4-39fcabe4d681]
protocols             : [OpenFlow13]
rstp_enable           : false
rstp_status           : {}
sflow                 : []
status                : {}
stp_enable            : false
```

Figura 40. Controlador Ryu asignado a Open vSwitch

Con todo lo anterior se puede afirmar que el controlador SDN Ryu se ha asignado correctamente al switch virtual Open vSwitch y está listo para su ejecución.

5.1.3 Configuración de las Raspberry Pi

Una vez realizados los pasos anteriores, será necesario probar el correcto funcionamiento de nuestra red SDN. Para ello, utilizaremos dos Raspberry Pi: la, tal y como se muestra en la **Figura 9. Esquema del proyecto**. La elección de las Raspberry Pi no es casual como bien se explica en el apartado **4.3.4 Dispositivos de red – Raspberry Pi**, su bajo coste y alta eficiencia permiten simular nodos finales en la red SDN, con el fin de generar tráfico de datos y simular diversos escenarios de red

Para garantizar un análisis detallado y preciso de nuestra SDN, es esencial una correcta instalación y configuración de las Raspberry Pi. Este proceso incluye la preparación del sistema operativo, la configuración de las interfaces de red y la instalación de los paquetes necesarios para la comunicación con el controlador Ryu y el switch virtual Open vSwitch.

Seguiremos la guía oficial de Raspberry Pi para la correcta instalación del sistema operativo [38]. Además, para agilizar el proceso, Raspberry Pi ha creado la herramienta Raspberry Pi Imager, mediante la cual se puede instalar la última versión disponible del sistema operativo Raspberry Pi OS de forma automática, tal como se indica en su web.

Una vez instalada la herramienta Raspberry Pi Imager, procederemos a su ejecución:

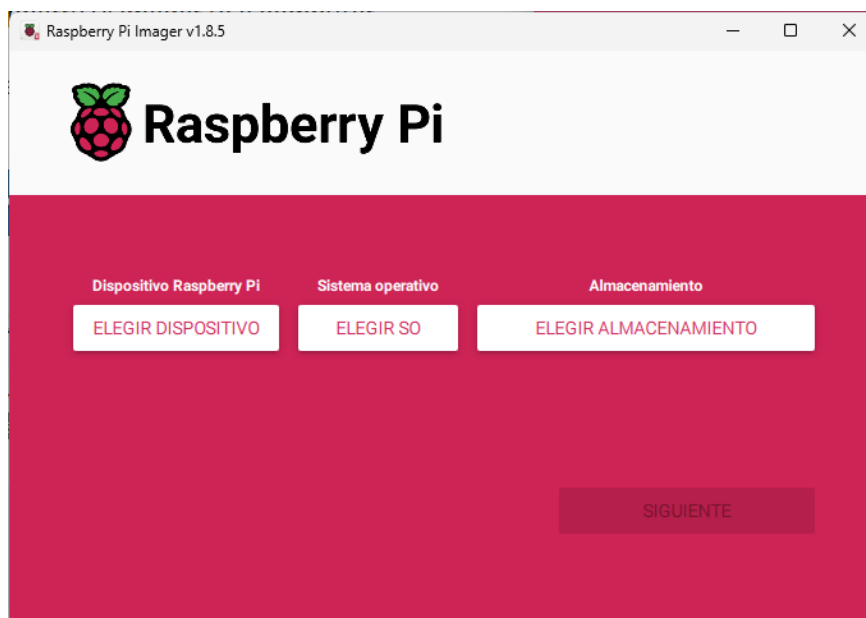


Figura 41. Herramienta Raspberry Pi Imager

En este caso se necesitará instalar el sistema operativo de Raspberry Pi en ambos dispositivos, por lo que se tendrá que seleccionar la versión correcta para cada una de las Raspberry Pi e introducir las tarjetas SD donde se instalará el sistema operativo.

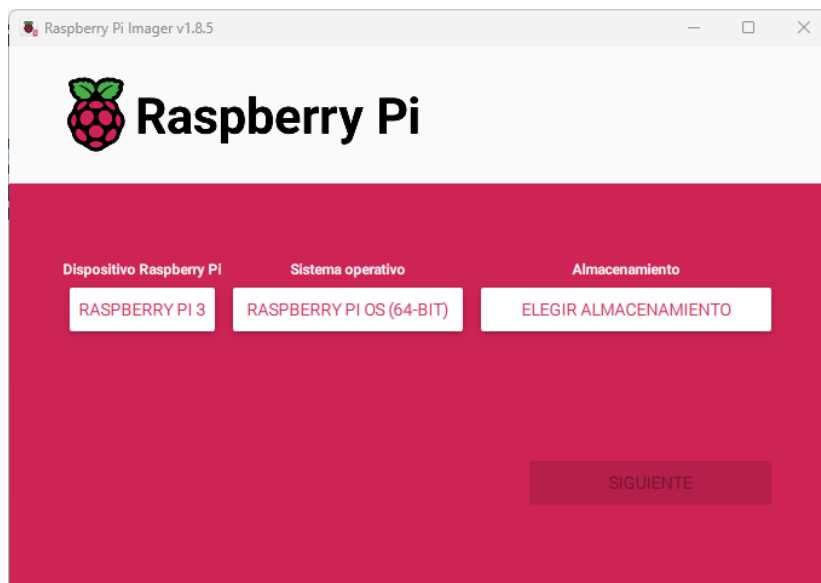


Figura 42. Instalación del S.O en Raspberry Pi 3 (I)

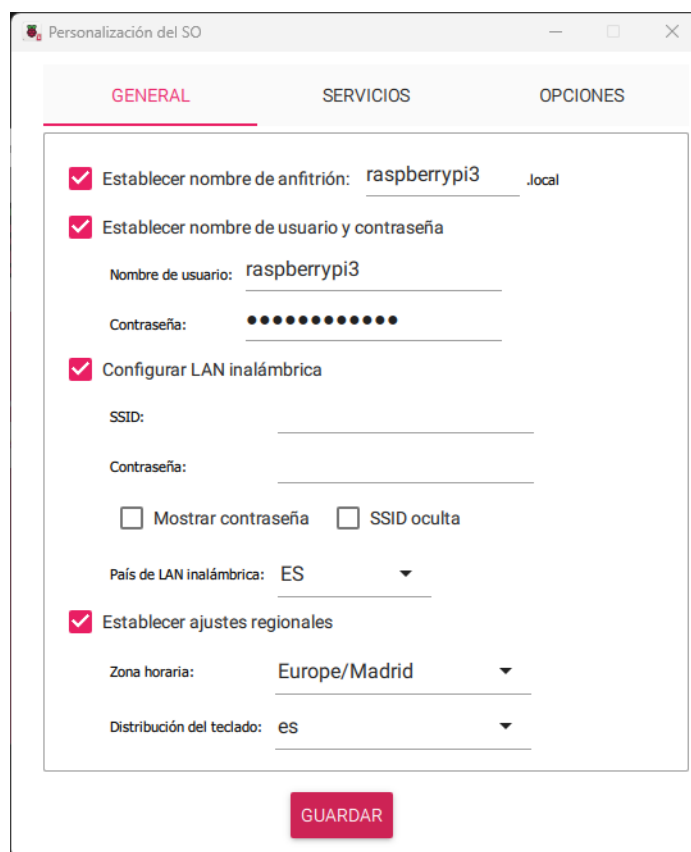


Figura 43. Instalación del S.O en Raspberry Pi 3 (II)

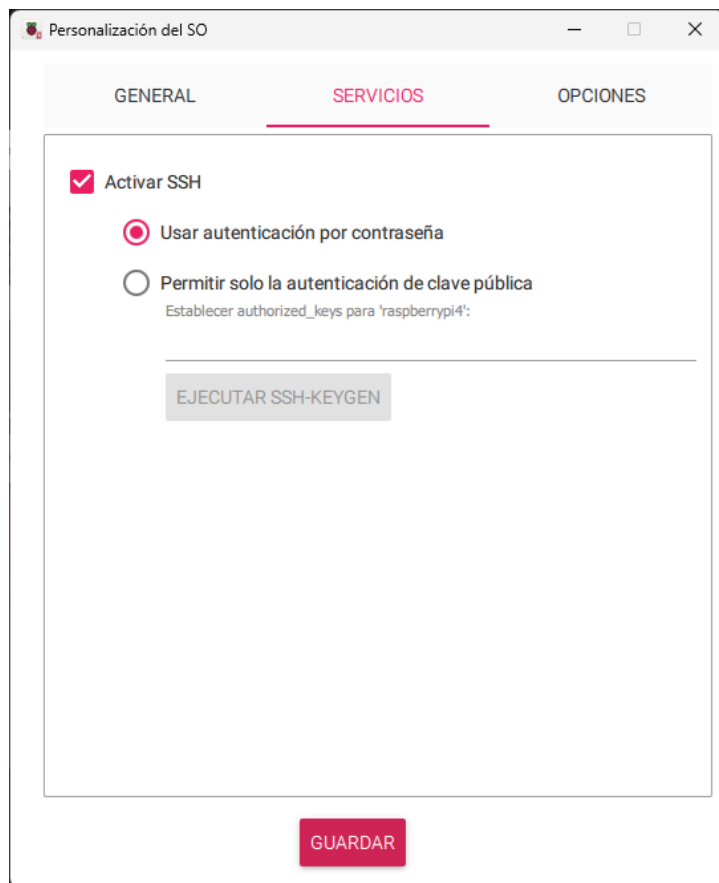


Figura 44. Instalación del S.O en Raspberry Pi 3 (III)

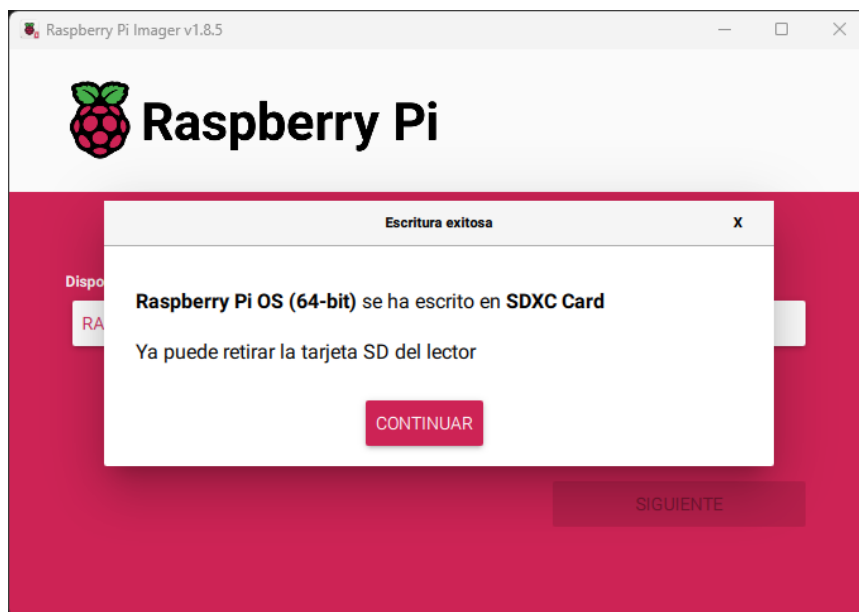


Figura 45. Instalación del S.O en Raspberry Pi 3 (IV)

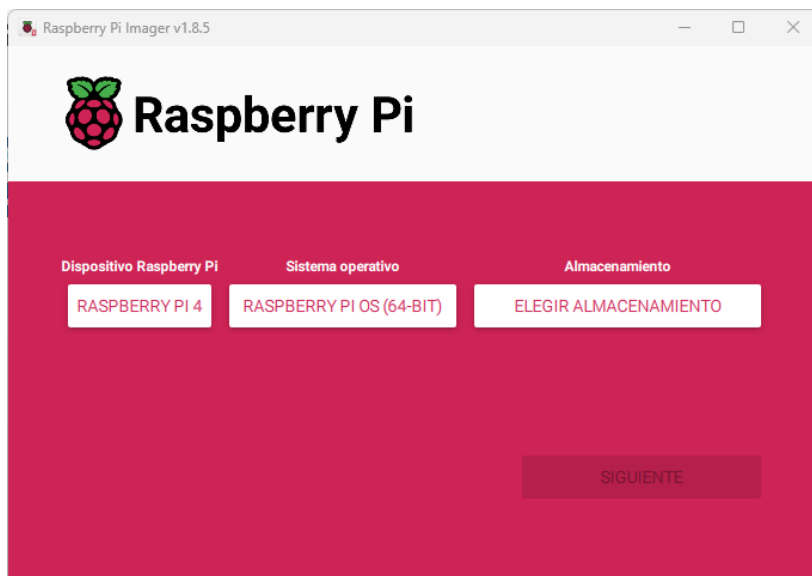


Figura 46. Instalación del S.O en Raspberry Pi 4 (I)

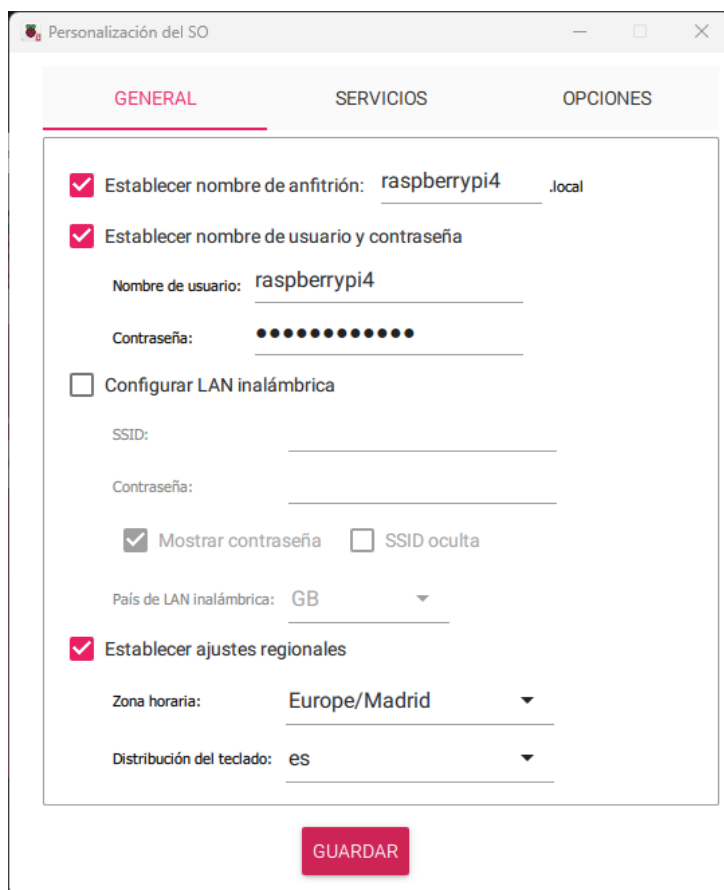


Figura 47. Instalación del S.O en Raspberry Pi 4 (II)

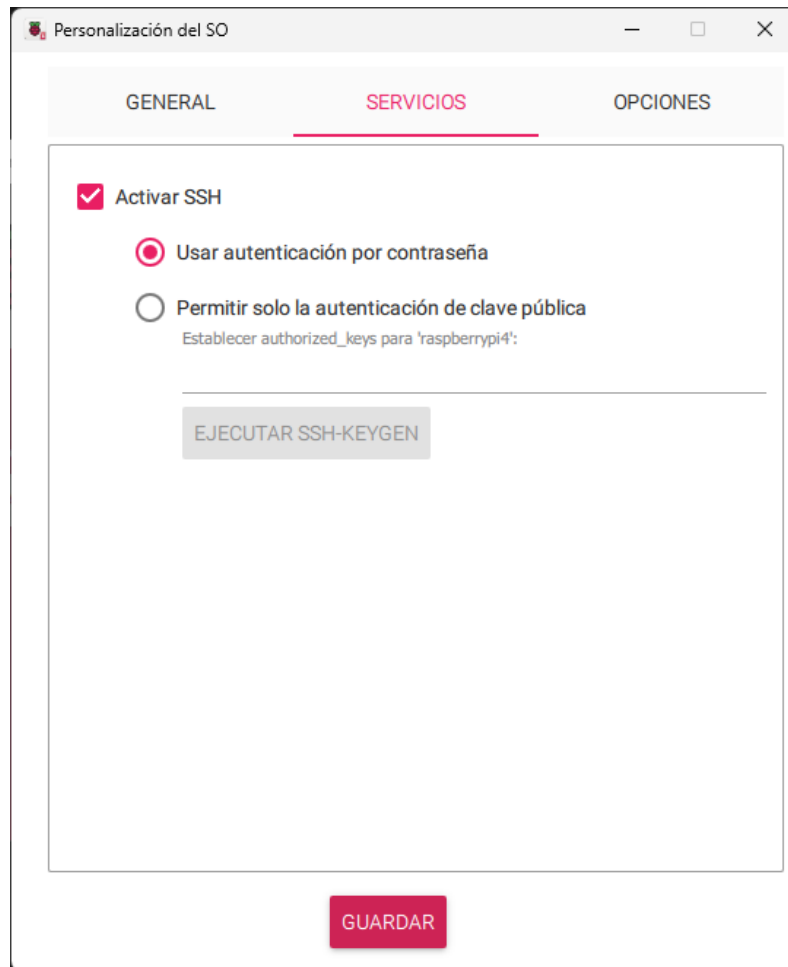


Figura 48. Instalación del S.O en Raspberry Pi 4 (III)

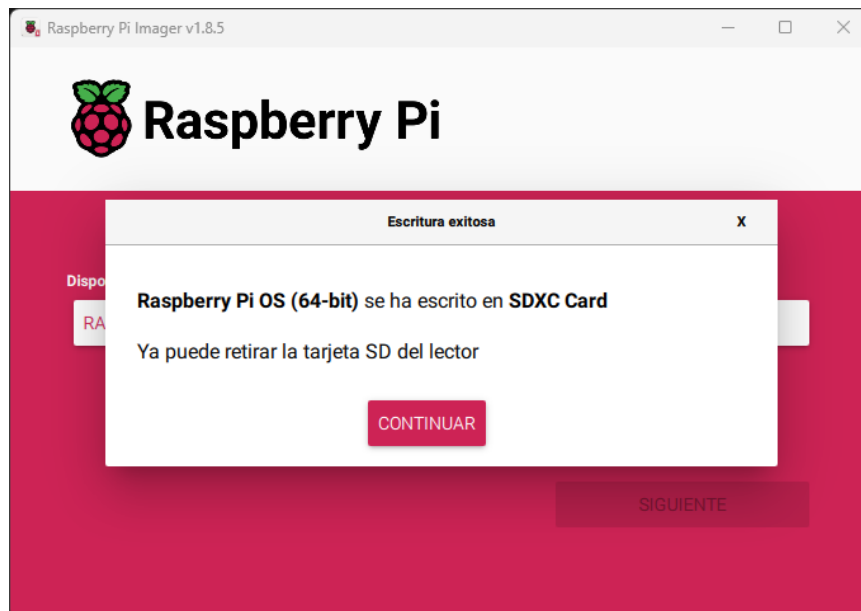


Figura 49. Instalación del S.O en Raspberry Pi 4 (IV)

Como se puede observar, durante el proceso de instalación se ha asignado el nombre de **raspberrypi3** y **raspberrypi4** a los dispositivos para poder identificarlos más fácilmente. Además, se ha habilitado la opción de conexión mediante SSH para poder acceder a las Raspberry Pi mediante el terminal.

Con esto los dispositivos aparecerán en la red con la siguiente configuración:

Dispositivos conectados (2,4G)	
raspberrypi3	
Dirección IP: 192.168.31.220	Dirección MAC: B8:27:EB:01:B0:7C

Dispositivos conectados(5G)	
raspberrypi4	
Dirección IP: 192.168.31.87	Dirección MAC: E4:5F:01:0F:68:D1

Figura 50. Raspberry Pi 3 y 4 conectadas a la red

Para añadir las Raspberry Pi a la red 10.0.0.X/24 como se ha hecho con las interfaces de red del switch virtual Open vSwitch en el apartado **5.1.1 Instalación y configuración de Open vSwitch**, se deberán asignar unas direcciones IP estáticas tanto a la interfaz **eth0** de la Raspberry Pi 3 como a la interfaz **eth0** de las Raspberry Pi 4. Para ello se deberá editar el archivo ubicado en `/etc/network/interfaces` con el siguiente comando:

```
sudo nano /etc/network/interfaces
```

Figura 51. Comando de acceso al archivo interfaces de las Raspberry Pi

Y se deberá añadir las siguientes líneas:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
#CONFIGURACION IP ESTATICA RASPERRY PI 3
source /etc/network/interfaces.d/*
auto eth0
iface eth0 inet static
    address 10.0.0.3
    netmask 255.255.255.0
    gateway 10.0.0.1
```

Figura 52. Configuración IP estática en Raspberry Pi 3


```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*
#CONFIGURACION IP ESTATICA RASPBERRY PI 4
auto eth0
iface eth0 inet static
    address 10.0.0.4
    netmask 255.255.255.0
    gateway 10.0.0.2
```

Figura 53. Configuración IP estática en Raspberry Pi 4

De este modo, tras reiniciar ambos dispositivos, se asignarán las IP 10.0.0.3 a la Raspberry Pi 3 y 10.0.0.4 a la Raspberry Pi 4. Las puertas de enlace del switch virtual serán 10.0.0.1 y 10.0.0.2, respectivamente.

```
raspberrypi3@raspberrypi3:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.3 netmask 255.255.255.0 broadcast 10.0.0.255
    inet6 fe80::6238:ce45:a97:c43d prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:54:e5:29 txqueuelen 1000 (Ethernet)
    RX packets 91 bytes 10698 (10.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 31 bytes 3982 (3.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 54. Configuración de red Raspberry Pi 3

```
raspberrypi4@raspberrypi4:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.4 netmask 255.255.255.0 broadcast 10.0.0.255
    inet6 fe80::c4a2:3203:c643:5b46 prefixlen 64 scopeid 0x20<link>
    ether e4:5f:01:0f:68:d0 txqueuelen 1000 (Ethernet)
    RX packets 229 bytes 39522 (38.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 41 bytes 4085 (3.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 55. Configuración de red Raspberry Pi 4

5.1.4 Creación del puente virtual br0

Por último, cuando se hayan realizado todos los pasos anteriores, es decir, se haya instalado y configurado Open vSwitch en una máquina Linux, se haya instalado las dependencias y todo lo necesario para poder instalar y configurar correctamente el controlador Ryu, y las Raspberry Pi estén encendidas y correctamente configuradas las interfaces de red, se podrá proceder a la realización de un bridge virtual.

Un puente (bridge) en Open vSwitch se utiliza principalmente para crear una red virtual que conecta varias interfaces de red, permitiendo que se comuniquen entre sí como si estuvieran en la misma red física. En el caso de este proyecto, el uso Open vSwitch, junto con un controlador SDN como Ryu, permite la gestión centralizada de la red.

La creación de un bridge **br0** nos permitirá facilitar la implementación de políticas de red avanzadas, la optimización del tráfico y la creación de topologías de red flexibles con las que se podrá analizar los distintos casos prácticos que se han planteado en el proyecto.

En primer lugar, lo más necesario será modificar el siguiente fichero:

```
sudo nano /etc/network/interfaces
```

Figura 56. Modificación del archivo interfaces del sistema

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*

auto enp0s8
iface enp0s8 inet manual

auto enp0s9
iface enp0s9 inet manual

#Bride br0
auto br0
iface br0 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    bridge_ports enp0s8 enp0s9
# IP secundaria para br0
iface br0 inet static
    address 10.0.0.2
    netmask 255.255.255.0
```

Figura 57. Archivo interfaces modificado

Tras modificar el archivo **interfaces** como se muestra en la **Figura 57. Archivo interfaces modificado**, se configura varias interfaces de red en un sistema. Primero, establece la interfaz de loopback (**lo**), que permite que el sistema se comunique consigo mismo, configurándola para que se inicie automáticamente con el método loopback. Luego, configura dos interfaces físicas (**enp0s8** y **enp0s9**) para que también se inicien automáticamente, pero sin asignarles una dirección IP directamente, ya que se utilizarán en el puente de red utilizado para nuestro entorno. A continuación, se configura una interfaz de puente (**br0**), que incluye estas dos interfaces físicas, permitiendo que actúen como puertos en una red local. A esta interfaz de puente **br0** se le asigna una dirección IP estática **10.0.0.1** con una máscara de subred **255.255.255.0**. Además, se añade una segunda dirección IP estática **10.0.0.2** a la misma interfaz **br0**, también en la misma subred. En resumen, esta configuración permite que el sistema funcione como un puente de red con dos interfaces físicas y que tenga dos direcciones IP en la misma red local.

Para conseguir crear un bridge en Open vSwitch seguiremos la documentación oficial [29], la cual indica que se deben ejecutar los siguientes comandos para la creación de un puente virtual:

```
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 enp0s8
sudo ovs-vsctl add-port br0 enp0s9
sudo ovs-vsctl set bridge br0 protocols=openFlow13
sudo ip addr add 10.0.0.1/24 dev br0
sudo ip addr add 10.0.0.2/24 dev br0
sudo ip link set br0 up
```

Figura 58. Comandos para la creación del puente (bridge) br0

Con estos comandos, se crea un puente llamado **br0** en Open vSwitch, agregando las interfaces de red del switch virtual Open vSwitch **enp0s8** y **enp0s9** al puente, además se establece el protocolo OpenFlow 1.3 que se va a utilizar. Tras esto, se asignan las direcciones IP 10.0.0.1 y 10.0.0.2 al puente **br0**. Finalmente, se activa el puente **br0** para que esté listo para enrutar el tráfico entre los dispositivos de la red.

Para comprobar que el puente virtual **br0** ha sido creado correctamente podemos ejecutar este comando para ver si las direcciones IP han sido asignadas a las interfaces especificadas:

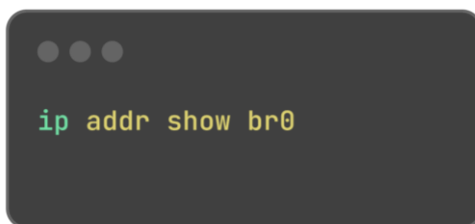


Figura 59. Verificación del puente br0

```
openvswitch@openvswitch:~/Escritorio$ ip addr show br0
6: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 08:00:27:c3:87:32 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global br0
        valid_lft forever preferred_lft forever
    inet 169.254.218.196/16 brd 169.254.255.255 scope global noprefixroute br0
        valid_lft forever preferred_lft forever
    inet 10.0.0.2/24 brd 10.0.0.255 scope global secondary br0
        valid_lft forever preferred_lft forever
    inet6 fe80::2b1c:477a:bfbf:657a/64 scope link
        valid_lft forever preferred_lft forever
```

Figura 60. Configuración puente br0

En este caso tanto la interfaz **enp0s8** como **enp0s9** tienen asignadas correctamente las direcciones 10.0.0.1/24 y 10.0.0.2/24 respectivamente.

5.2 Diseño y análisis de distintos casos prácticos

Como recordatorio, la **Figura 9. Esquema del proyecto** del apartado **5.1 Preparación del entorno práctico**, muestra el esquema general del proyecto en el que se fundamentan los distintos casos prácticos desarrollados en este capítulo. Este esquema proporciona una visión integral de la estructura y conexión entre los dispositivos, facilitando la comprensión de los objetivos y el enfoque de cada caso práctico. En el desarrollo que sigue, cada caso práctico tomará como referencia este esquema inicial, adaptándose a distintas configuraciones y modos de operación. A continuación, se muestra nuevamente la **Figura 9** para reforzar esta referencia visual:

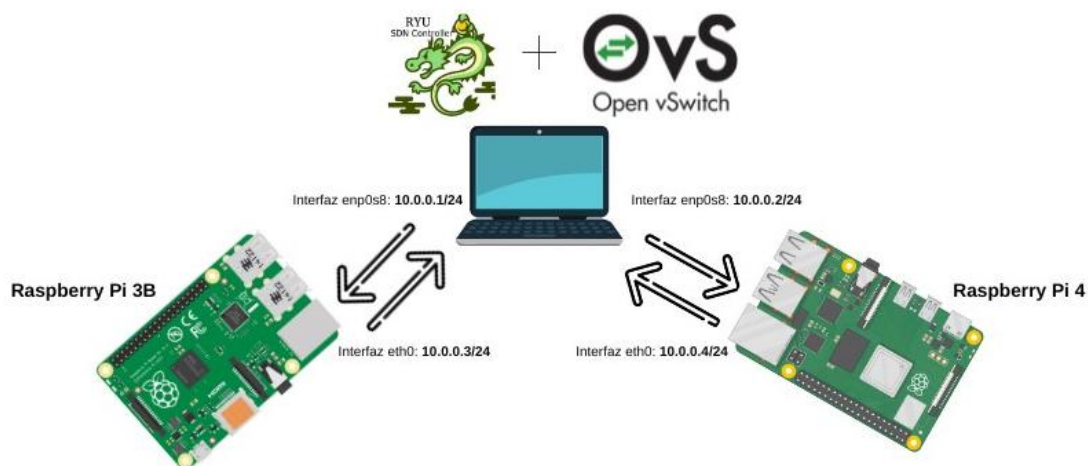


Figura 61. Esquema del proyecto

5.2.1 Caso 1: Conexión entre los dispositivos – Visibilidad de la red

Como se menciona en el apartado **4.3.2.1 Reglas de flujo**, las reglas de flujo en Open vSwitch son cruciales para la gestión eficiente de las redes SDN y constituyen la base que permite a estas redes ser más dinámicas y programables, adaptándose de manera ágil y efectiva a las necesidades.

Por ello una vez realizados todos los pasos de instalación y configuración del apartado **5.1 Preparación del entorno práctico**, se deben crear reglas de flujo con el fin de crear una serie de instrucciones para determinar cómo deben tratarse los paquetes que atraviesan el switch virtual Open vSwitch.

En este primer caso práctico, se pretende conectar todos los dispositivos de red entre sí, para que haya visibilidad entre todos ellos. Antes que nada, aclarar que se podría llegar a tener conexión entre los dispositivos de nuestro entorno gracias a Open vSwitch, y el script que utilizemos de Ryu para ejecutar el controlador, ya que se podrían meter en el propio script sin necesidad de introducirlas manualmente, pero en este caso en concreto utilizaremos la conexión mediante rutas estáticas gracias a la creación de reglas de flujo. Para tener este resultado, se deben aplicar las siguientes reglas de flujo, siguiendo la documentación oficial de Open vSwitch [29]:

```
sudo ovs-ofctl -O OpenFlow13 add-flow br0
"priority=1,in_port=br0,actions=output:enp0s8,enp0s9"
sudo ovs-ofctl -O OpenFlow13 add-flow br0
"priority=1,in_port=enp0s8,actions=output:br0"
sudo ovs-ofctl -O OpenFlow13 add-flow br0
"priority=1,in_port=enp0s9,actions=output:br0"
```

Figura 62. Creación de las reglas de flujo para conectar todos los dispositivos

- La primer regla de flujo configurada con el comando **sudo ovs-ofctl -O OpenFlow13 add-flow br0 "priority=1,in_port=br0,actions=output:enp0s8,enp0s9"** establece que todos los paquetes que ingresen por el puerto **br0** (el puente de Open vSwitch) serán enviados a los puertos **enp0s8** y **enp0s9**. Esta regla tiene una prioridad de 1, lo que significa que se evaluará después de las reglas con una prioridad más alta (número mayor). La acción especificada en esta regla garantiza que los paquetes coincidentes se reenvíen simultáneamente a ambos puertos **enp0s8** y **enp0s9**, asegurando una distribución dual del tráfico.
- La segunda regla de flujo, configurada con el comando **sudo ovs-ofctl -O OpenFlow13 add-flow br0 "priority=1,in_port=enp0s8,actions=output:br0"**, tiene una prioridad de 1 y se aplica a los paquetes que ingresan por el puerto **enp0s8**. La acción especificada es enviar estos paquetes al puerto **br0**, asegurando que todos los paquetes que entren por **enp0s8** sean reenviados al puente **br0**.
- La tercera regla de flujo, configurada con el comando **sudo ovs-ofctl -O OpenFlow13 add-flow br0 "priority=1,in_port=enp0s9,actions=output:br0"**, también tiene una prioridad de 1 y se aplica a los paquetes que ingresan por el puerto **enp0s9**. La acción para los paquetes coincidentes es enviarlos al puerto **br0**, de modo que todos los paquetes que ingresen por **enp0s9** serán enviados al puente **br0**.

Para asegurarnos de que todos los dispositivos e interfaces de red están conectados correctamente y verificar que las reglas de flujo creadas funcionan correctamente, utilizamos la herramienta ping. El ping es una utilidad sencilla pero muy útil que nos permite verificar la conectividad entre dos dispositivos en una red. Básicamente, lo que hace es enviar un mensaje desde un dispositivo a otro y esperar una respuesta.

Este mensaje utiliza un protocolo llamado ICMP (Protocolo de Mensajes de Control de Internet) [50]. Cuando enviamos un ping, el dispositivo de destino recibe un paquete de solicitud y, si todo está bien, responde con un paquete de respuesta. Esto nos confirma que la comunicación entre los dispositivos es posible y también nos permite medir el tiempo que tarda en completarse este intercambio de mensajes, ayudándonos a identificar posibles problemas en la red.

Para comprobar que la conexión entre los dispositivos es correcta lanzamos el siguiente ping:

```
for ip in 10.0.0.1 10.0.0.2 10.0.0.3 10.0.0.4; do ping -c 3 $ip; done
```

Figura 63. Ping de comprobación de conexión entre dispositivos

Tras ejecutar este comando desde todos los dispositivos de red (Open vSwitch, Raspberry Pi 3 y Raspberry Pi 4), se recibe la siguiente respuesta:

```
openvswitch@openvswitch:~$ for ip in 10.0.0.1 10.0.0.2 10.0.0.3 10.0.0.4; do ping -c 3 $ip; done
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.049 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.172 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.032/0.084/0.172/0.062 ms
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=0.028 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.033 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.028/0.031/0.034/0.002 ms
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=1.85 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.47 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=1.43 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.425/1.582/1.851/0.190 ms
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=1.52 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.977 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.819 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.819/1.104/1.517/0.298 ms
```

Figura 64. Respuesta al ping desde Open vSwitch

```
raspberrypi3@raspberrypi3:~$ for ip in 10.0.0.1 10.0.0.2 10.0.0.3 10.0.0.4; do ping -c 3 $ip; done
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=2.96 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.55 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=1.30 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.295/1.934/2.957/0.730 ms
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=4.38 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.36 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1.24 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.239/2.325/4.375/1.449 ms
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=0.116 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.128 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.131 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.116/0.125/0.131/0.006 ms
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=3.80 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=2.10 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=2.03 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 2.034/2.643/3.800/0.818 ms
```

Figura 65. Respuesta al ping desde Raspberry Pi 3

```
raspberrypi4@raspberrypi4:~$ for ip in 10.0.0.1 10.0.0.2 10.0.0.3 10.0.0.4; do ping -c 3 $ip; done
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1.93 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.03 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=1.01 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.007/1.323/1.930/0.429 ms
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=1.16 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.02 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1.04 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.023/1.075/1.162/0.061 ms
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=3.91 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=2.21 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=2.33 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 2.210/2.816/3.912/0.776 ms
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.143 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.072 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2044ms
rtt min/avg/max/mdev = 0.072/0.101/0.143/0.030 ms
```

Figura 66. Respuesta al ping desde Raspberry Pi 4

Como se puede observar la conexión entre todos los dispositivos es correcta y las reglas de flujo funcionan correctamente, con esto el entorno ya está listo para realizar casos prácticos más complejos necesarios para el análisis del funcionamiento de la red SDN.

5.2.1.1 Reconexión de un dispositivo

Como se indica en la documentación oficial de Open vSwitch [29], en el caso de que un dispositivo se desconecte del switch debido a un fallo de red, Open vSwitch tiene la capacidad de recuperar la conexión automáticamente. A continuación, se describe un ejemplo práctico de cómo se realiza este proceso de reconexión.

Se detecta que un dispositivo, por ejemplo, **enp0s8**, se ha desconectado debido a un fallo de red. Open vSwitch registra la pérdida de conexión y se prepara para reestablecerla automáticamente.

Open vSwitch utiliza las reglas de flujo preconfiguradas para intentar reestablecer la conexión con el dispositivo desconectado. Las reglas configuradas permiten que, una vez el dispositivo se reconecte físicamente, los paquetes se manejen adecuadamente sin necesidad de reconfiguración manual.

Suponiendo que el dispositivo **enp0s8** se ha reconectado físicamente, Open vSwitch automáticamente aplicará las reglas de flujo para reestablecer la comunicación. Las reglas configuradas para este puerto, tal como se mencionó anteriormente, se asegurarán de reenviar los paquetes al puente br0 y viceversa.

Para verificar que el dispositivo **enp0s8** se ha reconectado correctamente y que las reglas de flujo están funcionando, se puede utilizar el comando ping para comprobar la conectividad entre los dispositivos. Por ejemplo: ping **10.0.0.3**, donde **10.0.0.3** es la IP del dispositivo reconectado. Tras ejecutar este comando desde el Open vSwitch y otros dispositivos de la red, se espera una respuesta positiva que confirme la reconexión.

```
openvswitch@openvswitch:~$ ping 10.0.0.3
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=5.04 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=3.08 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=1.83 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=1.29 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=1.29 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=14.4 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=1.47 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=1.27 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=3.03 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=1.48 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=1.26 ms
64 bytes from 10.0.0.3: icmp_seq=12 ttl=64 time=4.49 ms
64 bytes from 10.0.0.3: icmp_seq=26 ttl=64 time=16.4 ms
64 bytes from 10.0.0.3: icmp_seq=27 ttl=64 time=1.97 ms
64 bytes from 10.0.0.3: icmp_seq=28 ttl=64 time=1.17 ms
64 bytes from 10.0.0.3: icmp_seq=29 ttl=64 time=5.13 ms
64 bytes from 10.0.0.3: icmp_seq=30 ttl=64 time=4.28 ms
64 bytes from 10.0.0.3: icmp_seq=31 ttl=64 time=1.38 ms
64 bytes from 10.0.0.3: icmp_seq=32 ttl=64 time=1.31 ms
64 bytes from 10.0.0.3: icmp_seq=33 ttl=64 time=1.35 ms
64 bytes from 10.0.0.3: icmp_seq=34 ttl=64 time=1.62 ms
^C
--- 10.0.0.3 ping statistics ---
34 packets transmitted, 21 received, 38.2353% packet loss, time 34270ms
rtt min/avg/max/mdev = 1.170/3.548/16.379/4.070 ms
```

Figura 67. Reconexión automática de dispositivo

Después de la reconexión, el dispositivo debería tener comunicación con todos los dispositivos que tenía conexión antes del fallo en la red.

Este proceso de reconexión demuestra la capacidad de Open vSwitch para gestionar fallos de red y reestablecer conexiones automáticamente, manteniendo la estabilidad y continuidad del servicio en una red SDN.

5.2.2 Caso 2: Configuración y Pruebas de Control de Tráfico y Calidad de Servicio (QoS)

Este caso práctico del proyecto se enfocará en la configuración y el estudio de dos aspectos clave: la Calidad de Servicio (QoS) y la limitación de tráfico entrante (*ingress policing*). Estos mecanismos son cruciales para asegurar un rendimiento de red eficiente y evitar la congestión.

El contenido se divide en tres apartados principales. Primero, se abordará la configuración de QoS para controlar el tráfico saliente en interfaces específicas. A continuación, se establecerán reglas de *policing* para gestionar el tráfico entrante. Por último, se realizarán pruebas de ancho de banda y rendimiento de red para verificar que las configuraciones implementadas funcionan correctamente. Estas pruebas son esenciales para validar la efectividad de las configuraciones y asegurar que los recursos de red se utilizan de manera óptima.

Como ya hemos tratado en el apartado **4.4 Ventajas y desventajas del uso de SDN frente a las redes tradicionales**, SDN permite una mayor flexibilidad y control centralizado sobre la red, facilitando la gestión y configuración dinámica de los recursos de red. Con este enfoque, es posible responder rápidamente a cambios en el tráfico y necesidades de los usuarios, mejorar la eficiencia operativa y reducir los costes asociados. Este caso práctico ayudará a verificar si las redes SDN realmente proporcionan estos beneficios, demostrando su superioridad en términos de gestión de tráfico y optimización de rendimiento.

En este caso práctico se utilizará un script modificado por mí, aunque creado por otro usuario de la amplia comunidad de desarrolladores que tiene Ryu [51], aunque también se podría usar cualquier script por defecto que trae Ryu donde detalla el funcionamiento del controlador. En este caso se ha utilizado este porque las pruebas han resultado bastante satisfactorias en cuanto al tema de interconexión de dispositivos etc.

El script que hemos utilizado configura un switch OpenFlow 1.3 usando el controlador Ryu, diseñado para manejar el tráfico de red de manera eficiente aprendiendo direcciones MAC. Al iniciar, instala una regla por defecto que envía todos los paquetes no coincidentes al controlador. Cuando se recibe un paquete, el switch examina sus direcciones MAC de origen y destino. Si la dirección de destino es conocida, envía el paquete al puerto correspondiente; de lo contrario, difunde el paquete. Además, aprende la asociación de direcciones MAC a puertos para evitar la difusión en el futuro. Como hemos mencionado en el apartado **5.2.1 Caso 1: Conexión entre los dispositivos – Visibilidad de la red**, también existe la posibilidad de agregar reglas de flujo dinámicamente, que es justo lo que hace este script, y esto sirve para gestionar paquetes similares sin necesidad de enviarlos al controlador, optimizando así el rendimiento de la red.

El script utilizado se puede observar en las siguientes páginas:

```
prueba_qos_v3_1.py

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                         ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                           actions)]

        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)

        datapath.send_msg(mod)
```

Figura 68. Script de Ryu para las pruebas de QoS (I)

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    if eth.ethertype == ether_types.ETH_TYPE_LLDP:
        return
    dst = eth.dst
    src = eth.src

    dpid = format(datapath.id, "d").zfill(16)
    self.mac_to_port.setdefault(dpid, {})

    self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

    # learn a mac address to avoid FLOOD next time.
    self.mac_to_port[dpid][src] = in_port

    if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [parser.OFPActionOutput(out_port)]

    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            self.add_flow(datapath, 1, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow(datapath, 1, match, actions)
    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

Figura 69. Script de Ryu para las pruebas de QoS (II)

```
openvswitch@openvswitch:~/Escritorio/Ryu_scripts$ ryu-manager --verbose prueba_qos_v3_1.py
loading app prueba_qos_v3_1.py
loading app ryu.controller.ofp_handler
instantiating app prueba_qos_v3_1.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f7041c1c610> address:('127
.0.0.1', 53588)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f7041c8b760>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xca21ab2e,OFPSwitchFeatures(a
uxiliary_id=0,capabilities=79,datapath_id=8796760147762,n_buffers=0,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 0008796760147762 f0:a7:31:2a:46:4a ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 0008796760147762 f0:a7:31:2a:46:4a ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 0008796760147762 b8:27:eb:54:e5:29 08:00:27:c3:87:32 1
```

Figura 70. Ryu lanzado utilizando el script modificado

Una vez ejecutado Ryu, se puede observar que se han generado las reglas de flujo siguientes mediante el siguiente comando.

```
prueba_qos_v3_1.py
sudo ovs-ofctl -O OpenFlow13 dump-flows br0
```

Figura 71. Comando para consultar las reglas de flujo creadas

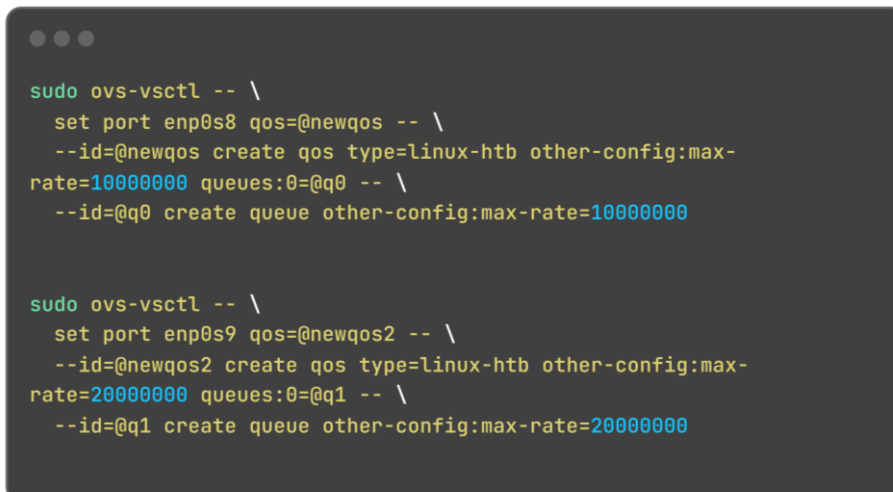
```
prueba_qos_v3_1.py
cookie=0x0, duration=66.169s, table=0, n_packets=0, n_bytes=0,
priority=1,in_port=LOCAL,dL_src=08:00:27:c3:87:32,dL_dst=b8:27:eb:54:e5:29
actions=output:enp0s8
cookie=0x0, duration=61.130s, table=0, n_packets=2, n_bytes=180,
priority=1,in_port=enp0s8,dL_src=b8:27:eb:54:e5:29,dL_dst=08:00:27:c3:87:32 actions=LOCAL
cookie=0x0, duration=40.344s, table=0, n_packets=5, n_bytes=390,
priority=1,in_port=enp0s9,dL_src=e4:5f:01:0f:68:d0,dL_dst=08:00:27:c3:87:32 actions=LOCAL
cookie=0x0, duration=35.192s, table=0, n_packets=1, n_bytes=42,
priority=1,in_port=LOCAL,dL_src=08:00:27:c3:87:32,dL_dst=e4:5f:01:0f:68:d0
actions=output:enp0s9
cookie=0x0, duration=80.034s, table=0, n_packets=20, n_bytes=3712, priority=0
actions=CONTROLLER:65535
```

Figura 72. Reglas de flujo creadas por el script

5.2.2.1 Creación y configuración de QoS

Con todo lo anterior, para empezar con la configuración de la Calidad de Servicio (**QoS**) en las interfaces de red, seguiremos la documentación oficial de Open vSwitch [52]. En primer lugar, es necesario definir límites de ancho de banda y colas de tráfico para controlar cómo se distribuye el tráfico saliente. Open vSwitch (OVS) permite realizar estas configuraciones utilizando el algoritmo **Hierarchical Token Bucket** (HTB), que facilita la gestión y priorización del tráfico.

A continuación, se presentan los comandos necesarios para configurar la QoS en dos interfaces de red específicas (**enp0s8** y **enp0s9**). Estos comandos establecen una tasa máxima de transferencia de datos y crean colas para gestionar el tráfico de manera eficiente.



```
sudo ovs-vsctl -- \
  set port enp0s8 qos=@newqos -- \
  --id=@newqos create qos type=linux-htb other-config:max-
rate=100000000000000 queues:0=@q0 -- \
  --id=@q0 create queue other-config:max-rate=100000000

sudo ovs-vsctl -- \
  set port enp0s9 qos=@newqos2 -- \
  --id=@newqos2 create qos type=linux-htb other-config:max-
rate=200000000000000 queues:0=@q1 -- \
  --id=@q1 create queue other-config:max-rate=200000000
```

Figura 73. Configuración de la QoS

En estos comandos:

- Se configura el puerto enp0s8 para usar una nueva política de QoS denominada @**newqos**, con una tasa máxima de **10 Mbps**.
- Se crea una cola asociada @**q0** con la misma tasa máxima de **10 Mbps**.
- De manera similar, se configura el puerto enp0s9 para usar una política de QoS denominada @**newqos2**, con una tasa máxima de **20 Mbps**.
- Se crea una cola asociada @**q1** con una tasa máxima de **20 Mbps**.

Estas configuraciones garantizan que el tráfico saliente en cada puerto no exceda las tasas especificadas, optimizando así el uso de los recursos de red y asegurando una distribución equitativa del ancho de banda.

5.2.2.2 Configuración de Ingress Policing

Una vez configurada la **QoS**, configuraremos la limitación de tráfico entrante (*ingress policing*) en las interfaces de red siguiendo la documentación oficial de Open vSwitch [52]. La técnica para controlar la cantidad de tráfico que una interfaz de red puede recibir se conoce como *Ingress Policing*, esto permite evitar la congestión y asegurar que el tráfico entrante no supere una tasa especificada. Configuraremos las tasas de policing y las ráfagas permitidas en las interfaces `enp0s8` y `enp0s9` para gestionar eficientemente el tráfico entrante.

Para `enp0s8`, estableceremos una tasa de policing de 10,000 kbps (**10 Mbps**), lo que significa que la interfaz no permitirá recibir más de 10 megabits de datos por segundo. Además, configuraremos una ráfaga permitida de 8,000 kbps, que define la cantidad máxima de datos que puede ser procesada temporalmente durante un corto periodo antes de que se aplique la limitación. De manera similar, para `enp0s9`, la tasa de policing será de 20,000 kbps (**20 Mbps**), y la ráfaga permitida será de 16,000 kbps. La ráfaga permitida (definida en kilobits por segundo) es menor que la tasa de policing (también en kilobits por segundo) porque se refiere a la capacidad de manejar picos de tráfico a corto plazo sin rechazar paquetes inmediatamente.

Los comandos específicos para esta configuración son los siguientes:

```
sudo ovs-vsctl set interface enp0s8 ingress_policing_rate=10000
sudo ovs-vsctl set interface enp0s8 ingress_policing_burst=8000

sudo ovs-vsctl set interface enp0s9 ingress_policing_rate=20000
sudo ovs-vsctl set interface enp0s9 ingress_policing_burst=16000
```

Figura 74. Creación de Ingress Policing Rate y Burst

Para comprobar que la QoS se ha creado en el apartado anterior, así como las limitaciones de tráfico para las interfaces elegidas, usaremos los siguientes comandos:

```
sudo ovs-vsctl list qos
sudo ovs-vsctl list queue
```

Figura 75. Comprobación de QoS y Ingress Policing

La ejecución de los comandos anteriores, debería dar un resultado como este:

```
openvswitch@openvswitch:~$ sudo ovs-vsctl list qos
_uuid          : 4780bc35-fac3-4292-b955-edb104bfe736
external_ids   : {}
other_config   : {max-rate="10000000"}
queues        : {0=3907dbac-faba-46be-b301-4771a16b599a}
type           : linux-htb

_uuid          : 1aed6d76-b69f-4b74-a688-c86459939110
external_ids   : {}
other_config   : {max-rate="20000000"}
queues        : {0=fc63b4e1-ab76-4e76-8b17-049743de0c41}
type           : linux-htb
openvswitch@openvswitch:~$ sudo ovs-vsctl list queue
_uuid          : 3907dbac-faba-46be-b301-4771a16b599a
dscp           : []
external_ids   : {}
other_config   : {max-rate="10000000"}

_uuid          : fc63b4e1-ab76-4e76-8b17-049743de0c41
dscp           : []
external_ids   : {}
other_config   : {max-rate="20000000"}
```

Figura 76. QoS e Ingress Policing creadas correctamente

Como se puede observar en la **Figura 76. QoS e Ingress Policing creadas correctamente**, tanto la QoS como las políticas de *Ingress Policing* han sido creadas correctamente con los ajustes establecidos.

Para verificar que las QoS y las políticas de *Ingress Policing* se han asociado a las interfaces especificadas, se deben ejecutar los siguientes comandos:

```
sudo ovs-vsctl list port enp0s8
sudo ovs-vsctl list port enp0s9
```

Figura 77. Interfaces asociadas a QoS e Ingress Policing

La ejecución de estos comandos debería devolver un resultado como el siguiente, donde se puede observar que la QoS ha sido asignada a las interfaces especificadas:


```
openvswitch@openvswitch:~$ sudo ovs-vsctl list port enp0s8
_uuid                : 28b61757-ee62-44ac-a620-9e99e6de3f6a
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [ab1f3518-dedb-4bd1-90e2-502ac5c242c5]
lacp                : []
mac                 : []
name                : enp0s8
other_config        : {}
protected           : false
qos                 : 4780bc35-fac3-4292-b955-edb104bfe736
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
```

Figura 78. Configuración interfaz enp0s8

```
openvswitch@openvswitch:~$ sudo ovs-vsctl list port enp0s9
[sudo] contraseña para openvswitch:
_uuid                : 53e4f340-09cf-4de1-97a4-39fcabe4d681
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [e0385cd2-3126-487f-89e9-0deada7f9c03]
lacp                : []
mac                 : []
name                : enp0s9
other_config        : {}
protected           : false
qos                 : 1aed6d76-b69f-4b74-a688-c86459939110
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
```

Figura 79. Configuración interfaz enp0s9

5.2.2.3 Verificación de ancho de banda y rendimiento de red

En este apartado, se realizará la verificación del ancho de banda y del rendimiento de la red para asegurar que las configuraciones de QoS y policing implementadas en las interfaces de red están funcionando correctamente.

Las pruebas de ancho de banda son esenciales para validar que los límites de tráfico establecidos están siendo respetados y que la red está operando de manera eficiente sin congestión. Se utilizarán herramientas como `iperf3` para medir el rendimiento de la red [53], tanto en términos de capacidad máxima de transferencia de datos como en la respuesta a picos de tráfico. Estas pruebas proporcionarán una visión clara del comportamiento de la red bajo diferentes condiciones de carga, permitiendo identificar y corregir posibles problemas de configuración o rendimiento.

5.2.2.3.1 Verificación con Políticas y QoS Establecidas

En este subapartado, se comprobará cómo funciona la red con las políticas de QoS y las configuraciones de policing que ya se establecieron en las interfaces `enp0s8` y `enp0s9`. Para ello, se realizarán pruebas de ancho de banda que nos permitirán verificar si el tráfico de datos se mantiene dentro de los límites configurados y si se está gestionando de manera eficiente. Utilizaremos el siguiente comando para realizar las pruebas de ancho de banda:

```
En el servidor:  
iperf3 -s -p 5001  
  
En el cliente:  
iperf3 -c 10.0.0.1 -t 60 -p 5001
```

Figura 80. Comandos de `iperf3` para verificar el rendimiento

Estos comandos permiten iniciar una prueba de ancho de banda entre un cliente y un servidor, conectándose a la dirección `10.0.0.1` perteneciente a una de las interfaces de la máquina que ejecuta Open vSwitch, durante 60 segundos, utilizando el puerto 5001. Con esta prueba se confirmará que las tasas de tráfico no superan los límites configurados y que el tráfico se gestiona de manera eficiente, garantizando un rendimiento óptimo de la red.

- **Cliente Raspberry Pi 3 (10.0.0.3):**

```

raspberrypi3@raspberrypi3:~ $ iperf3 -c 10.0.0.1 -t 60 -p 5001
Connecting to host 10.0.0.1, port 5001
[ 5] local 10.0.0.3 port 48466 connected to 10.0.0.1 port 5001
[ ID] Interval           Transfer             Bitrate             Retr    Cwnd
[ 5]  0.00-1.00    sec    2.56 MBytes        21.4 Mbits/sec      132    4.24 KBytes
[ 5]  1.00-2.00    sec    1018 KBytes        8.34 Mbits/sec      138    4.24 KBytes
[ 5]  2.00-3.00    sec    1.24 MBytes        10.4 Mbits/sec      106    4.24 KBytes
[ 5]  3.00-4.00    sec    1.12 MBytes        9.38 Mbits/sec      123    2.83 KBytes
[ 5]  4.00-5.00    sec    1.24 MBytes        10.4 Mbits/sec      111    19.8 KBytes
[ 5]  5.00-6.00    sec    1.24 MBytes        10.4 Mbits/sec      125    2.83 KBytes
[ 5]  6.00-7.00    sec    1.24 MBytes        10.4 Mbits/sec      101    5.66 KBytes
[ 5]  7.00-8.00    sec    954 KBytes         7.82 Mbits/sec     126    2.83 KBytes
[ 5]  8.00-9.00    sec    1.43 MBytes        12.0 Mbits/sec     104    5.66 KBytes
[ 5]  9.00-10.00   sec    1.24 MBytes        10.4 Mbits/sec     101    7.07 KBytes
[ 5] 10.00-11.00   sec    1.18 MBytes        9.90 Mbits/sec      91    2.83 KBytes
[ 5] 11.00-12.00   sec    1.24 MBytes        10.4 Mbits/sec     126    4.24 KBytes
[ 5] 12.00-13.00   sec    1.18 MBytes        9.90 Mbits/sec     132    2.83 KBytes
[ 5] 13.00-14.00   sec    1018 KBytes        8.34 Mbits/sec     118    2.83 KBytes
[ 5] 14.00-15.00   sec    1.24 MBytes        10.4 Mbits/sec     128    7.07 KBytes
[ 5] 15.00-16.00   sec    1018 KBytes        8.34 Mbits/sec      90    5.66 KBytes
[ 5] 16.00-17.00   sec    1.24 MBytes        10.4 Mbits/sec     103    2.83 KBytes
[ 5] 17.00-18.00   sec    1.24 MBytes        10.4 Mbits/sec     122    4.24 KBytes
[ 5] 18.00-19.00   sec    1.24 MBytes        10.4 Mbits/sec     116    2.83 KBytes
[ 5] 19.00-20.00   sec    1.24 MBytes        10.4 Mbits/sec     113    4.24 KBytes
[ 5] 20.00-21.00   sec    954 KBytes         7.82 Mbits/sec     103    2.83 KBytes
[ 5] 21.00-22.00   sec    1.24 MBytes        10.4 Mbits/sec     111    2.83 KBytes
[ 5] 22.00-23.00   sec    1.18 MBytes        9.90 Mbits/sec     131    4.24 KBytes
[ 5] 23.00-24.00   sec    1.22 MBytes        10.2 Mbits/sec     124    7.07 KBytes
[ 5] 24.00-25.00   sec    1.39 MBytes        11.7 Mbits/sec     106    7.07 KBytes
[ 5] 25.00-26.00   sec    764 KBytes         6.25 Mbits/sec     100    1.41 KBytes
[ 5] 26.00-27.00   sec    1.49 MBytes        12.5 Mbits/sec     118    4.24 KBytes
[ 5] 27.00-28.00   sec    1.18 MBytes        9.90 Mbits/sec     146    4.24 KBytes
[ 5] 28.00-29.00   sec    1.24 MBytes        10.4 Mbits/sec     125    4.24 KBytes
[ 5] 29.00-30.00   sec    1.18 MBytes        9.90 Mbits/sec     106    4.24 KBytes
[ 5] 30.00-31.00   sec    1018 KBytes        8.34 Mbits/sec     124    5.66 KBytes
[ 5] 31.00-32.00   sec    1.18 MBytes        9.90 Mbits/sec     100    2.83 KBytes

```

Figura 81. Ejecución de iperf3 en Raspberry Pi 3 (10.0.0.3)

A partir de esta imagen de las pruebas iperf3 ejecutadas desde la Raspberry Pi 3 con IP 10.0.0.3, podemos extraer varias conclusiones sobre el rendimiento de la red y la efectividad de las configuraciones de QoS y policing implementadas en la interfaz enp0s8:

- **Tasa de Transferencia:** La tasa de transferencia varía entre 6.25 Mbits/sec y 21.4 Mbits/sec, con la mayoría de las mediciones alrededor de los 8-10.4 Mbits/sec.
- **Promedio de Bitrate:** La tasa de bitrate promedio es de aproximadamente 10.1 Mbits/sec en el sender y 9.99 Mbits/sec en el receiver, lo que sugiere que la tasa de policing configurada de 10 Mbps en enp0s8 está siendo respetada en general.
- **Retransmisiones (Retr):** Hay un número significativo de retransmisiones (6922 en total), lo que indica que el tráfico excedente está siendo eliminado o que hay congestión en la red.

- Cliente Raspberry Pi 4 (10.0.0.4):

```
raspberrypi4@raspberrypi4:~$ iperf3 -c 10.0.0.1 -t 60 -p 5001
Connecting to host 10.0.0.1, port 5001
[ 5] local 10.0.0.4 port 49586 connected to 10.0.0.1 port 5001
[ ID] Interval          Transfer          Bitrate          Retr          Cwnd
[ 5] 0.00-1.00        sec 5.07 MBytes     42.5 Mbits/sec   374          25.5 KBytes
[ 5] 1.00-2.00        sec 2.42 MBytes     20.3 Mbits/sec   251          39.6 KBytes
[ 5] 2.00-3.00        sec 2.30 MBytes     19.3 Mbits/sec   234          24.0 KBytes
[ 5] 3.00-4.00        sec 1.93 MBytes     16.2 Mbits/sec   211          9.90 KBytes
[ 5] 4.00-5.00        sec 2.36 MBytes     19.8 Mbits/sec   227          8.48 KBytes
[ 5] 5.00-6.00        sec 2.73 MBytes     22.9 Mbits/sec   195          15.6 KBytes
[ 5] 6.00-7.00        sec 2.49 MBytes     20.9 Mbits/sec   247          4.24 KBytes
[ 5] 7.00-8.00        sec 2.05 MBytes     17.2 Mbits/sec   223          8.48 KBytes
[ 5] 8.00-9.00        sec 2.24 MBytes     18.8 Mbits/sec   245          12.7 KBytes
[ 5] 9.00-10.00       sec 2.42 MBytes     20.3 Mbits/sec   230          9.90 KBytes
[ 5] 10.00-11.00      sec 2.24 MBytes     18.8 Mbits/sec   260          7.07 KBytes
[ 5] 11.00-12.00     sec 3.04 MBytes     25.5 Mbits/sec   238          11.3 KBytes
[ 5] 12.00-13.00     sec 2.30 MBytes     19.3 Mbits/sec   246          4.24 KBytes
[ 5] 13.00-14.00     sec 2.42 MBytes     20.3 Mbits/sec   177          8.48 KBytes
[ 5] 14.00-15.00     sec 1.99 MBytes     16.7 Mbits/sec   218          4.24 KBytes
[ 5] 15.00-16.00     sec 2.24 MBytes     18.8 Mbits/sec   215          14.1 KBytes
[ 5] 16.00-17.00     sec 2.49 MBytes     20.9 Mbits/sec   233          2.83 KBytes
[ 5] 17.00-18.00     sec 2.49 MBytes     20.9 Mbits/sec   263          7.07 KBytes
[ 5] 18.00-19.00     sec 2.36 MBytes     19.8 Mbits/sec   217          4.24 KBytes
[ 5] 19.00-20.00     sec 2.24 MBytes     18.8 Mbits/sec   186          7.07 KBytes
[ 5] 20.00-21.00     sec 2.30 MBytes     19.3 Mbits/sec   201          2.83 KBytes
[ 5] 21.00-22.00     sec 2.67 MBytes     22.4 Mbits/sec   274          17.0 KBytes
[ 5] 22.00-23.00     sec 2.30 MBytes     19.3 Mbits/sec   224          35.4 KBytes
[ 5] 23.00-24.00     sec 2.05 MBytes     17.2 Mbits/sec   232          12.7 KBytes
[ 5] 24.00-25.00     sec 2.73 MBytes     22.9 Mbits/sec   180          4.24 KBytes
[ 5] 25.00-26.00     sec 2.42 MBytes     20.3 Mbits/sec   270          73.5 KBytes
[ 5] 26.00-27.00     sec 2.49 MBytes     20.9 Mbits/sec   216          7.07 KBytes
[ 5] 27.00-28.00     sec 1.99 MBytes     16.7 Mbits/sec   239          4.24 KBytes
[ 5] 28.00-29.00     sec 2.30 MBytes     19.3 Mbits/sec   236          1.41 KBytes
[ 5] 29.00-30.00     sec 2.73 MBytes     22.9 Mbits/sec   249          33.9 KBytes
[ 5] 30.00-31.00     sec 2.36 MBytes     19.8 Mbits/sec   218          8.48 KBytes
```

Figura 82. Ejecución de iperf3 en Raspberry Pi 4 (10.0.0.4)

A partir de la salida completa del iperf3 para la interfaz **enp0s9** ejecutada desde la Raspberry Pi 4 con IP 10.0.0.4, podemos extraer varias conclusiones sobre el rendimiento de la red y la efectividad de las configuraciones de QoS y policing implementadas.:

- **Tasa de Transferencia:** La tasa de transferencia varía entre 15.6 Mbits/sec y 42.5 Mbits/sec, con la mayoría de las mediciones alrededor de los 16-25 Mbits/sec.
- **Promedio de Bitrate:** La tasa de bitrate promedio es de aproximadamente 20.2 Mbits/sec tanto para el sender como para el receiver, lo que sugiere que la tasa de policing configurada de 20 Mbps en enp0s9 está siendo respetada en general.
- **Retransmisiones (Retr):** Hay un número significativo de retransmisiones (13,694 en total), lo que indica que el tráfico excedente está siendo eliminado o que hay congestión en la red.

La configuración de QoS parece estar funcionando correctamente al mantener la tasa de transferencia promedio en torno a los 20 Mbps, lo que coincide con la tasa de policing establecida. Sin embargo, las retransmisiones altas indican que puede haber picos en el tráfico que causan pérdidas de paquetes y la necesidad de retransmisión.

5.2.2.3.2 Verificación de Ingress Policing y Eliminación de Paquetes

En este subapartado, se especificará la configuración del ingress policing para observar si los paquetes se eliminan cuando las tasas de tráfico entrante exceden los límites configurados. Se configurarán tasas más pequeñas en enp0s8 para inducir la eliminación de paquetes y verificar el comportamiento de la red bajo estas condiciones. Los comandos para ajustar las tasas de policing son:

```
sudo ovs-vsctl set interface enp0s8 ingress_policing_rate=1000
sudo ovs-vsctl set interface enp0s8 ingress_policing_burst=100
```

Figura 83. Comandos para reajustar las policing establecidas y comprobar la eliminación de paquetes

Estas nuevas pruebas ayudarán a confirmar que la configuración de *Ingress policing* está funcionando como se espera, eliminando paquetes cuando el tráfico entrante excede los límites especificados y asegurando así la estabilidad y eficiencia de la red.

Para realizar las nuevas pruebas con las policing nuevas, ejecutaremos el siguiente comando, donde pondremos uno de los peores casos posibles:

```
iperf3 -c 10.0.0.3 -b 50M -p 5001 -t 60
```

Figura 84. Comando de iperf3 para comprobar la eliminación de paquetes

Este comando inicia una prueba de ancho de banda desde el cliente (la máquina que ejecuta el comando) hacia el servidor iperf3 en 10.0.0.3, utilizando un puerto específico (5001) y limitando la tasa de transferencia a 50 megabits por segundo durante 60 segundos. Además, al ser ejecutado desde la Raspberry Pi 4 con la IP 10.0.0.4 a la Raspberry Pi 3 con la IP 10.0.0.3, permitirá que el tráfico vaya desde la interfaz enp0s9 hasta la enp0s8 y en esta se apliquen las policing establecidas. La ejecución de este código debe devolver una respuesta como la siguiente:

```

raspberrypi4@raspberrypi4:~ $ iperf3 -c 10.0.0.3 -b 50M -p 5001 -t 60
Connecting to host 10.0.0.3, port 5001
[ 5] local 10.0.0.4 port 53458 connected to 10.0.0.3 port 5001
[ ID] Interval      Transfer      Bitrate      Retr  Cwnd
[ 5]  0.00-1.00    sec  1.25 MBytes  10.5 Mbits/sec  0   77.8 KBytes
[ 5]  1.00-2.00    sec  0.00 Bytes   0.00 bits/sec   2   1.41 KBytes
[ 5]  2.00-3.00    sec  0.00 Bytes   0.00 bits/sec   1   1.41 KBytes
[ 5]  3.00-4.00    sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5]  4.00-5.00    sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5]  5.00-6.00    sec  0.00 Bytes   0.00 bits/sec   1   1.41 KBytes
[ 5]  6.00-7.00    sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5]  7.00-8.00    sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5]  8.00-9.00    sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5]  9.00-10.00   sec  0.00 Bytes   0.00 bits/sec   1   1.41 KBytes
[ 5] 10.00-11.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 11.00-12.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 12.00-13.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 13.00-14.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 14.00-15.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 15.00-16.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 16.00-17.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 17.00-18.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 18.00-19.00   sec  0.00 Bytes   0.00 bits/sec   1   1.41 KBytes
[ 5] 19.00-20.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 20.00-21.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 21.00-22.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 22.00-23.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes
[ 5] 23.00-24.00   sec  0.00 Bytes   0.00 bits/sec   0   1.41 KBytes

```

Figura 85. Ejecución del comando anterior para verificar la eliminación de paquetes

A partir de la imagen anterior, donde se prueban las configuraciones de *Ingress policing* ajustadas, podemos extraer varias conclusiones sobre el rendimiento de la red y la efectividad de las configuraciones de *policing* para eliminar paquetes:

- **Tasa de Transferencia:** La tasa de transferencia se reduce drásticamente a partir del segundo segundo, manteniéndose en 0 bits/sec durante la mayor parte de la prueba.
- **Bitrate Promedio:** El bitrate promedio es extremadamente bajo, con solo 175 Kbits/sec transferidos en total por el sender y 0 bits/sec reportados por el receiver.
- **Retransmisiones (Retr):** Hay algunas retransmisiones (7 en total), pero la mayoría del tráfico parece ser completamente eliminado.

Estas observaciones indican claramente que la configuración de *Ingress policing* con una tasa de 1000 Kbits/sec (1 Mbps) y una ráfaga de 100 Kbits está funcionando como se esperaba. El tráfico excedente está siendo eliminado de manera efectiva, lo cual es evidente por la tasa de transferencia casi nula después del primer segundo.

Esta configuración asegura que cualquier tráfico entrante que exceda 1 Mbps sea eliminado, resultando en una transferencia de datos significativamente limitada y confirmando que el *Ingress policing* está operando de manera efectiva.

5.2.3 Caso 3: Configuración y uso de VLANs

En este apartado, abordaremos la configuración y uso de **VLANs** (Redes Virtuales de Área Local) en un entorno de redes definidas por software (SDN) utilizando Open vSwitch (OVS) y el controlador Ryu siguiendo la guía oficial de Open vSwitch donde se detalla la configuración de estas [54]. Las **VLANs** son esenciales para segmentar redes y mejorar tanto la seguridad como la eficiencia del tráfico de red. Este caso práctico se centrará en la implementación de **VLANs** en una red que incluye varias Raspberry Pi, proporcionando un ejemplo claro y detallado de cómo configurar y gestionar **VLANs** en un entorno SDN.

Describiremos el proceso de configuración de las interfaces de red en un switch virtual OVS, la asignación de estas interfaces a distintas **VLANs**, y cómo integrar todo esto con el controlador Ryu para gestionar la red de manera centralizada y dinámica.

Las **VLANs** (Redes Virtuales de Área Local) son una tecnología clave en la administración de redes, proporcionando diversas ventajas y aplicaciones en diferentes escenarios [55]. Algunos de los usos más comunes son:

- **Segmentación de Redes:** Las VLANs permiten dividir una red física en múltiples redes lógicas, lo que facilita la segmentación de tráfico. Esto es particularmente útil en grandes organizaciones donde se necesita separar el tráfico de diferentes departamentos, como finanzas, recursos humanos y desarrollo, mejorando la seguridad y la gestión del ancho de banda.
- **Seguridad:** Al segmentar una red mediante VLANs, se puede controlar mejor el acceso a recursos específicos. Por ejemplo, los usuarios de la VLAN de invitados pueden tener acceso restringido a ciertos servidores o aplicaciones, protegiendo así la información sensible de la organización.
- **Flexibilidad y Escalabilidad:** Las VLANs proporcionan flexibilidad al permitir la creación de redes lógicas que no están restringidas por la ubicación física de los dispositivos. Esto es útil en entornos donde los dispositivos pueden moverse o cuando se necesita reorganizar la red sin realizar cambios físicos en la infraestructura.
- **Aislamiento de Tráfico:** En escenarios donde se utilizan diferentes tipos de servicios o aplicaciones, las VLANs pueden aislar el tráfico de voz, video y datos, asegurando que cada tipo de tráfico reciba el tratamiento adecuado y no interfiera con otros tipos de tráfico.
- **Mejor seguridad en Redes SDN:** En redes definidas por software (SDN), las VLANs pueden ser utilizadas para segmentar y controlar el tráfico de manera más dinámica y eficiente. Los controladores SDN, como Ryu, pueden gestionar estas VLANs para adaptar la red a las necesidades cambiantes de forma automática y centralizada.

5.2.3.1 Tipos de Configuración de Puertos en VLAN

En la configuración de VLANs, existen dos tipos principales de puertos: puertos troncales (trunk) y puertos de acceso (access). La diferencia entre estos radica en cómo manejan las etiquetas VLAN en las tramas de red [56].

5.2.3.1.1 Puertos Troncales (Trunk)

Los puertos troncales están diseñados para transportar tráfico de múltiples VLANs. Estos puertos añaden y mantienen la etiqueta VLAN (información 802.1Q) en las tramas de red, esto es esencial cuando se conectan switches entre sí o cuando se conecta un switch a un dispositivo que necesita manejar múltiples VLANs, como un router o un controlador SDN. En una configuración trunk, la etiqueta VLAN es crucial para identificar a qué VLAN pertenece cada trama, permitiendo la segmentación adecuada del tráfico en el otro extremo del enlace trunk [57].

- **Ejemplo de configuración de un puerto trunk en Open vSwitch:**

```
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 enp0s8
```

Figura 86. Creación de un puerto troncal en Open vSwitch [57]

Este comando configura enp0s8 como un puerto trunk en el puente br0, permitiendo el paso de tráfico etiquetado con múltiples VLANs.

5.2.3.1.2 Puertos de Acceso (Access)

Por otro lado, los puertos de acceso están destinados a dispositivos finales que pertenecen a una única VLAN. Estos puertos no añaden etiquetas VLAN a las tramas de red, además las tramas que entran y salen por un puerto de acceso no llevan información de VLAN, lo que simplifica la comunicación con dispositivos que no necesitan entender las etiquetas VLAN. Un puerto de acceso se configura para pertenecer a una VLAN específica, asegurando que todo el tráfico que pasa por ese puerto es parte de esa VLAN sin necesidad de etiquetado adicional [57].

- **Ejemplo de configuración de un puerto de acceso en Open vSwitch:**

```
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 enp0s8 tag=1
```

Figura 87. Creación de un puerto de acceso en Open vSwitch [57]

Este comando configura enp0s8 como un puerto de acceso para VLAN 1 en el puente br0. Las tramas que pasan por enp0s8 no llevarán etiquetas VLAN, lo que facilita la conexión con dispositivos finales.

En resumen, los puertos troncales son utilizados para conexiones entre switches y otros dispositivos que manejan múltiples VLANs, manteniendo las etiquetas VLAN en las tramas. Los puertos de acceso están destinados a dispositivos finales en una única VLAN y no incluyen etiquetas VLAN en las tramas. Esta distinción es crucial para una correcta segmentación y manejo del tráfico en redes VLAN.

Una vez hemos comprendido la importancia del uso de VLANs en un entorno SDN así como los diferentes puertos que se utilizan para configurar las VLANs, vamos a proceder a su configuración en nuestro entorno de trabajo y a su posterior uso.

En este caso se ha decidido utilizar puertos troncales donde la asignación de la etiqueta se realiza en el propio script del controlador Ryu como se muestra en la **Figura 103. Reglas de flujo de simple_switch_13_vlan.py**, donde se crean las reglas de flujo con los identificadores de las VLANs deseadas.

En este caso práctico, todas las direcciones IP utilizadas pertenecen a la misma red, sin embargo, en una situación real, los dispositivos de diferentes VLANs estarían en redes con distintos rangos de direcciones. Esto se ha hecho intencionalmente para mostrar que, a pesar de compartir la misma red, los dispositivos en diferentes VLANs no pueden comunicarse entre sí.

Con esto y teniendo en cuenta nuestro entorno de trabajo, y siguiendo la guía oficial de Open vSwitch [54] [57], será necesario realizar los siguientes pasos para configurar correctamente las VLANs:

5.2.3.2 Creación de las VLANs para las Raspberry Pi

Para la creación de las VLANs, el primer comando agrega el puerto físico enp0s8 al switch virtual br0 en Open vSwitch. Este puerto se etiqueta con la VLAN 1, lo que significa que todo el tráfico que pasa por él se asociará a esta VLAN. De manera similar, el segundo comando añade el puerto físico enp0s9 al mismo switch virtual, pero esta vez con la etiqueta VLAN 2, asegurando que todo el tráfico de este puerto pertenezca a la VLAN 2. Cabe destacar que, aunque normalmente cada VLAN debería estar configurada con una red IP distinta (por ejemplo, VLAN 1 con 10.0.1.0/24 y VLAN 2 con 10.0.2.0/24), en este caso se ha utilizado intencionadamente la misma red IP (10.0.0.0/24) en ambas VLANs. Esto se hace como una prueba para demostrar que, a pesar de compartir la misma dirección IP en diferentes VLANs, no hay comunicación entre ellas sin un enrutamiento adecuado o reglas de flujo específicas. Se debe advertir que esta configuración no es recomendada en un entorno real, ya que podría generar problemas de red y confusión en la gestión del tráfico.

- **Creación de VLAN en Raspberry 10.0.0.3:**



```
sudo ovs-vsctl add-port br0 enp0s8
```

Figura 88. Asignación de interfaz enp0s8 a VLAN 1

- Creación de VLAN en Raspberry 10.0.0.4:

```
sudo ovs-vsctl add-port br0 enp0s9
```

Figura 89. Asignación de interfaz enp0s9 a VLAN 2

Con esta configuración realizada el esquema del entorno de trabajo sería el siguiente:

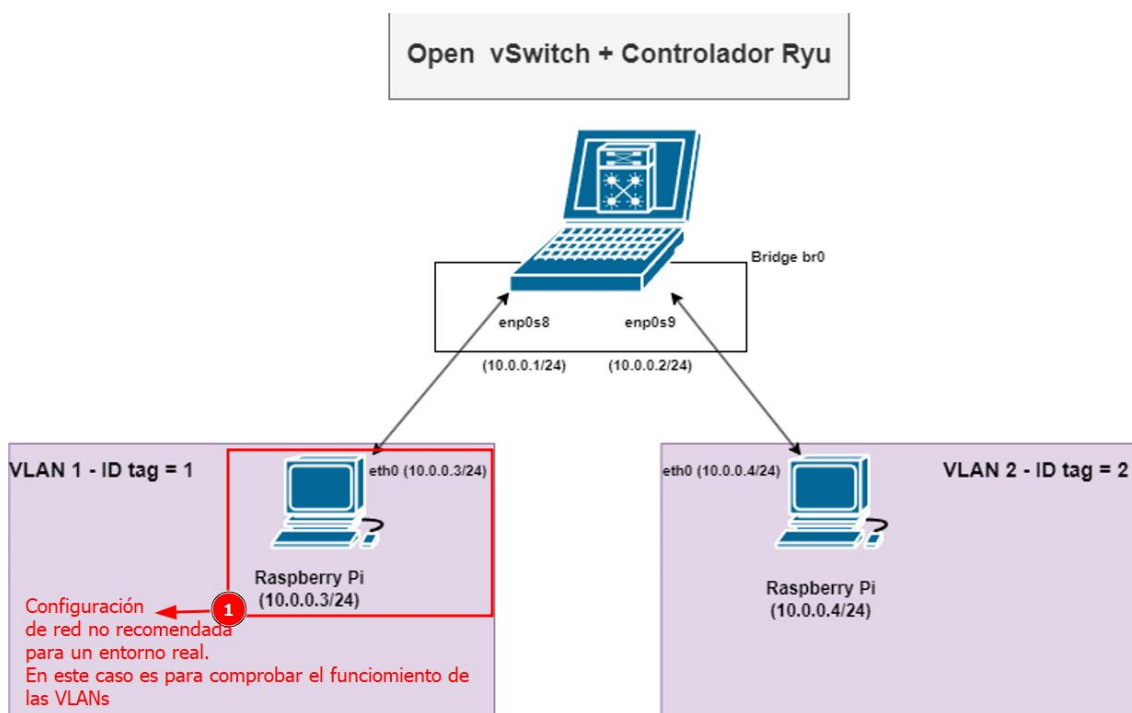


Figura 90. Entorno de trabajo VLAN 1 y 2 – 2 dispositivos

Una vez creadas las VLANs, es necesario modificar el script del controlador Ryu tal y como se hizo en el anterior apartado. En este caso es necesario modificar un script simple de los que nos proporciona Ryu en su repositorio oficial [46], pero en este caso es necesario importar algunos módulos adicionales de los que viene por defecto para poder utilizar correctamente las interfaces API como se explica en el apartado **4.2.1.1 Interfaces API**.

Al igual que en los casos anteriores, tomando como referencia el **script simple_switch_13.py** alojado en su repositorio oficial [58], y siguiendo la documentación oficial de Ryu para la configuración del controlador y la modificación de los scripts [35], y consultando posibles problemas y remediaciones de la amplia comunidad de Ryu [59], se ha elaborado el siguiente script para la interconexión de los dispositivos de la misma VLAN:

```
simple_switch_13_vlan.py

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet

class VLANIsolation(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(VLANIsolation, self).__init__(*args, **kwargs)

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Install table-miss flow entry
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

        # VLAN 1 - Permitir tráfico dentro de VLAN 1
        #Añadir reglas para permitir tráfico en la VLAN 1

        # VLAN 2 - Permitir tráfico dentro de VLAN 2
        #Añadir reglas para permitir tráfico en la VLAN 2

        # Bloquear tráfico entre VLAN 1 y VLAN 2
        #Añadir reglas para el bloqueo de tráfico entre VLAN 1 Y VLAN 2
```

Figura 91. simple_switch_13_vlan.py (I)

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
priority=priority,
                                match=match, instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)

def add_vlan_flow(self, datapath, vlan_id, in_port, out_port):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch(in_port=in_port, vlan_vid=(0x1000 | vlan_id))
    actions = [parser.OFPActionOutput(out_port)]
    self.add_flow(datapath, 1, match, actions)

def block_inter_vlan_traffic(self, datapath, in_port, out_port):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch(in_port=in_port)
    actions = []
    self.add_flow(datapath, 2, match, actions)
    self.logger.info("Bloqueando tráfico entre VLANs: in_port=%s, out_port=%s",
in_port, out_port)
```

Figura 92. simple_switch_13_vlan.py (II)

Como se puede observar, en el fragmento de script siguiente, se deben especificar las reglas de flujo y sus funciones, así como las etiquetas de las VLANs pertenecientes a cada una de las interfaces:

```
# Install table-miss flow entry
match = parser.OFPMatch()
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)

# VLAN 1 - Permitir tráfico dentro de VLAN 1
#Añadir reglas para permitir tráfico en la VLAN 1

# VLAN 2 - Permitir tráfico dentro de VLAN 2
#Añadir reglas para permitir tráfico en la VLAN 2

# Bloquear tráfico entre VLAN 1 y VLAN 2
#Añadir reglas para el bloqueo de tráfico entre VLAN 1 Y VLAN 2
```

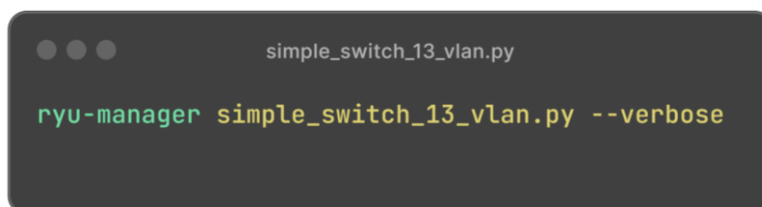
Figura 93. Reglas de flujo simple_switch_vlan.py

En este caso se ha fragmentado en tres partes:

- **Reglas Table-Miss:** Esta regla captura cualquier paquete que no coincida con ninguna otra regla y lo envía al controlador.
- **Reglas para VLAN 1:** Estas reglas permiten el tráfico dentro de VLAN 1 en los puertos enp0s10 y enp0s8.
- **Reglas para VLAN 2:** Estas reglas permiten el tráfico dentro de VLAN 2 en el puerto enp0s9.
- **Bloqueo de Tráfico entre VLANs:** Estas reglas bloquean el tráfico entre VLAN 1 y VLAN 2.

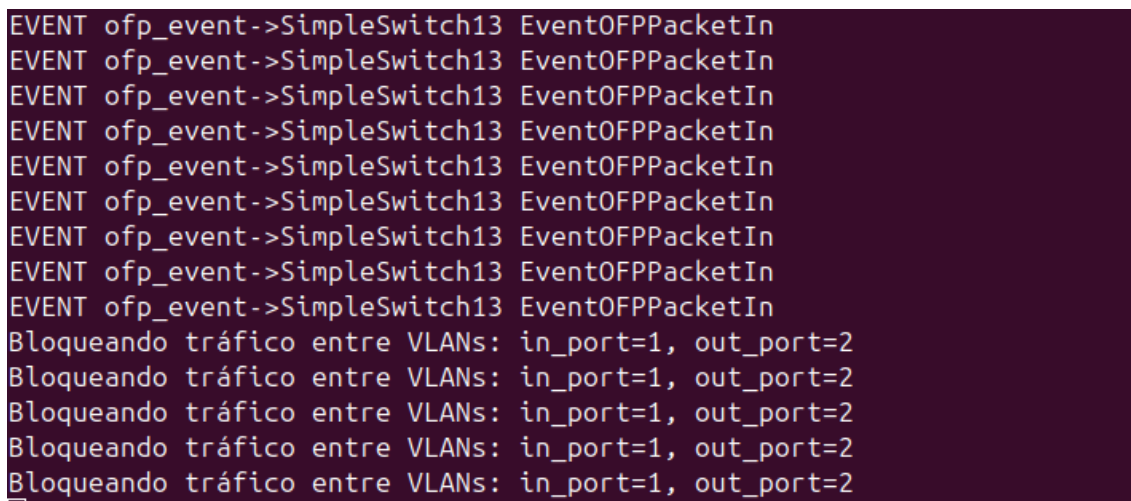
En este caso para evitar la redundancia de imágenes, se puede consultar un ejemplo de estas reglas en el apartado **5.2.3.3 Adición de otro dispositivo a la VLAN**.

Una vez terminado la modificación del script, es necesario ejecutarlo con el siguiente comando:



```
simple_switch_13_vlan.py
ryu-manager simple_switch_13_vlan.py --verbose
```

Figura 94. Ejecución del script simple_switch_13_vlan.py



```
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
Bloqueando tráfico entre VLANs: in_port=1, out_port=2
Bloqueando tráfico entre VLANs: in_port=1, out_port=2
Bloqueando tráfico entre VLANs: in_port=1, out_port=2
Bloqueando tráfico entre VLANs: in_port=1, out_port=2
Bloqueando tráfico entre VLANs: in_port=1, out_port=2
```

Figura 95. Log de Ryu tras ejecutar simple_switch_13_vlan.py

Como se observa en la imagen el controlador Ryu ya está bloqueando todo el tráfico entre VLAN 1 y VLAN 2. Para observar que está funcionando correctamente el etiquetado y que el entorno del proyecto está correctamente configurado, usamos Wireshark para observar el tráfico de la red.

En este caso no hay muchos dispositivos con lo que solo se puede observar el tráfico entre la Raspberry **10.0.0.3** a la interfaz **enp0s8 10.0.0.1**, así como de la Raspberry **10.0.0.4** a la interfaz **enp0s9 10.0.0.2**.

Tras capturar el tráfico de la red podemos observar que se está asignando correctamente la etiqueta a los paquetes procedentes de la VLAN 1, en este caso el tráfico va desde la Raspberry Pi 3 **10.0.0.3** a la interfaz **10.0.0.1** de Open vSwitch:

```
▼ Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits)
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun 21, 2024 19:08:30.577236000 Hora de verano romance
  UTC Arrival Time: Jun 21, 2024 17:08:30.577236000 UTC
  Epoch Arrival Time: 1718989710.577236000
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 0.000000000 seconds]
  [Time delta from previous displayed frame: 0.000000000 seconds]
  [Time since reference or first frame: 0.000000000 seconds]
  Frame Number: 1
  Frame Length: 46 bytes (368 bits)
  Capture Length: 46 bytes (368 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:vlan:ethertype:ip:icmp]
  [Coloring Rule Name: ICMP]
  [Coloring Rule String: icmp || icmpv6]
  ▼ Ethernet II, Src: RaspberryPiF_54:e5:29 (b8:27:eb:54:e5:29), Dst: PCSSystemtec_e1:6a:ff (08:00:27:e1:6a:ff)
    ▶ Destination: PCSSystemtec_e1:6a:ff (08:00:27:e1:6a:ff)
      ▶ Source: RaspberryPiF_54:e5:29 (b8:27:eb:54:e5:29)
        Type: 802.1Q Virtual LAN (0x8100)
      ▼ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 1
        000. .... .. = Priority: Best Effort (default) (0)
        ...0 .... .. = DEI: Ineligible
        ... 0000 0000 0001 = ID: 1
        Type: IPv4 (0x0800)
      ▶ Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.1
      ▶ Internet Control Message Protocol
```

Figura 96. Tráfico capturado - VLAN ID 1

Para comprobar que la VLAN funciona entre dispositivos de la misma VLAN, en el próximo apartado implementaremos nuevas reglas de flujo, y la adición de un nuevo dispositivo de red, una Raspberry Pi con la IP **10.0.0.5** conectada a una nueva interfaz de Open vSwitch **enp0s10** con la IP **10.0.0.6**.

5.2.3.3 Adición de otro dispositivo a la VLAN

Como se ha comentado en el apartado anterior, para comprobar que la VLAN funciona entre dispositivos de la misma VLAN, en este apartado implementaremos nuevas reglas de flujo y la adición de un nuevo dispositivo de red, una Raspberry Pi con la IP **10.0.0.5** conectada a una nueva interfaz de Open vSwitch **enp0s10** con la IP **10.0.0.6**, ahora procederemos a analizar el tráfico entre estos dispositivos para verificar la correcta configuración y funcionamiento de la VLAN.

```
raspberrypi5@raspberrypi5:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 10.0.0.5 netmask 255.255.255.0 broadcast 10.0.0.255
    ether 2c:cf:67:29:ba:54 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 106
```

Figura 97. Raspberry Pi con IP 10.0.0.5

Para añadir el dispositivo a la red SDN de nuestro entorno de trabajo y en concreto para que forme parte de la VLAN 1 de la red, es necesario añadir la interfaz conectada al nuevo dispositivo al puente virtual con el siguiente comando:

```
sudo ovs-vsctl add-port br0 enp0s10
```

Figura 98. Adición del nuevo dispositivo al puente virtual br0

Tras confirmar que ha sido correctamente añadida la nueva interfaz al puente virtual, es necesario seguir los puntos especificados en el apartado **5.1.4 Creación del puente virtual br0**.

Con esta configuración realizada el esquema del entorno de trabajo sería el siguiente:

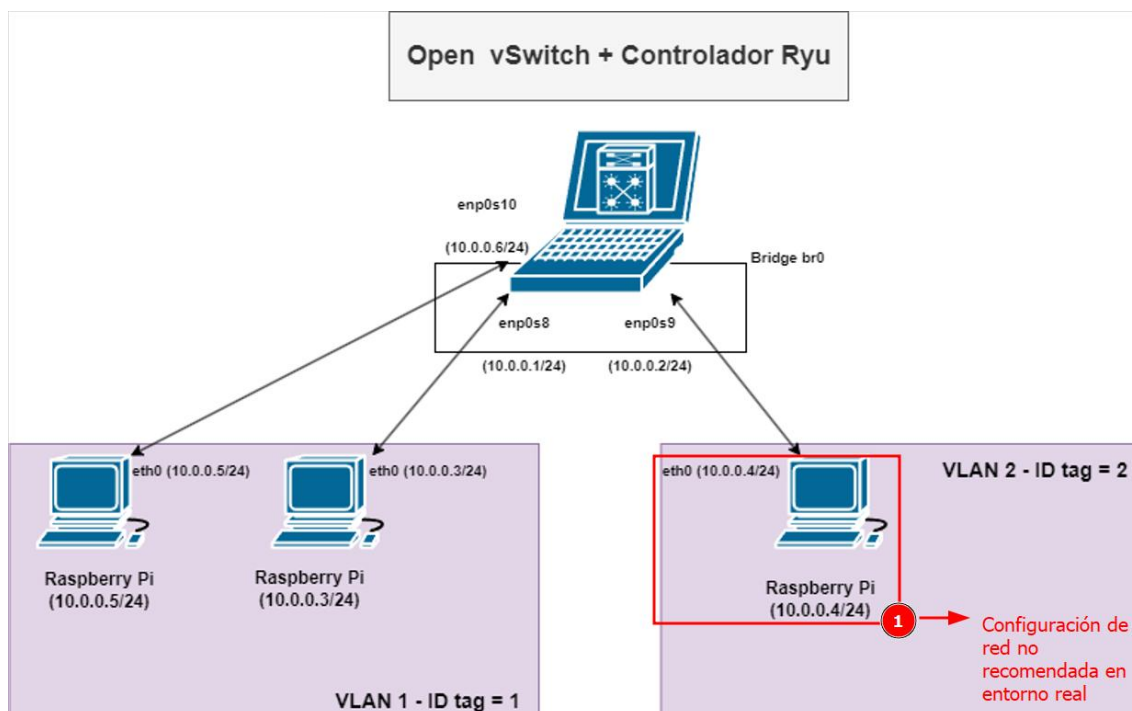


Figura 99. Entorno de trabajo VLAN 1 y 2 – 3 dispositivos

En Open vSwitch y el controlador Ryu, los puertos físicos como **enp0s8**, **enp0s9**, y **enp0s10** se identifican internamente con números de puerto. Cuando se configuran reglas de flujo en OpenFlow usando Ryu, se pueden usar estos números de puerto en lugar de los nombres de interfaz. Para consultar que números de puerto tienen asignados las interfaces se puede utilizar el siguiente comando:

```

  ● ● ●
  sudo ovs-ofctl show br0
  
```

Figura 100. Consulta de números de puerto de las interfaces del puente virtual br0

Una vez terminada esta configuración es necesario modificar las reglas de flujo del script utilizado en el apartado **5.2.3.2 Creación de las VLANs para las Raspberry Pi**. El apartado de las reglas de flujo sería el siguiente:

```

simple_switch_13_vlan.py

# Install table-miss flow entry
match = parser.OFPMatch()
actions = [parser.OFPACTIONOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)

# VLAN 1 - Permitir tráfico dentro de VLAN 1
self.add_vlan_flow(datapath, vlan_id=1, in_port=1, out_port=3)
self.add_vlan_flow(datapath, vlan_id=1, in_port=3, out_port=1)
self.add_vlan_flow(datapath, vlan_id=1, in_port=1, out_port=1)
self.add_vlan_flow(datapath, vlan_id=1, in_port=3, out_port=3)

# VLAN 2 - Permitir tráfico dentro de VLAN 2
self.add_vlan_flow(datapath, vlan_id=2, in_port=2, out_port=2)

# Bloquear tráfico entre VLAN 1 y VLAN 2
self.block_inter_vlan_traffic(datapath, in_port=1, out_port=2)
self.block_inter_vlan_traffic(datapath, in_port=2, out_port=1)
self.block_inter_vlan_traffic(datapath, in_port=3, out_port=2)
self.block_inter_vlan_traffic(datapath, in_port=2, out_port=3)

```

Figura 101. Reglas de flujo creadas en el script `simple_switch_13_vlan.py`

En el fragmento del script proporcionado, se especifican los **vlan_id** para las interfaces en las reglas de flujo que permiten el tráfico dentro de las VLANs. Para **VLAN 1**, se configuran las reglas de tráfico entre los **puertos 1** (correspondiente a la interfaz **enp0s8**) y **3** (correspondiente a la interfaz **enp0s10**). Para **VLAN 2**, se configuran las reglas de tráfico en el **puerto 2** (correspondiente a la interfaz **enp0s9**). Esto significa que cualquier paquete que entre por estos puertos con los **vlan_id** correspondientes será reenviado según las reglas definidas.

Tras añadir estas reglas se debe ejecutar el script con el siguiente comando:

```

simple_switch_13_vlan.py

ryu-manager simple_switch_13_vlan.py --verbose

```

Figura 102. Ejecución del script `simple_switch_13_vlan.py`

Si consultamos las reglas de flujo que crea el script para controlar el tráfico de la red, obtenemos las siguiente:

```
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0, priority=0
actions=CONTROLLER:65535
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=1, vlan_vid=0x1001 actions=output:3
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=3, vlan_vid=0x1001 actions=output:1
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=1, vlan_vid=0x1001 actions=output:1
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=3, vlan_vid=0x1001 actions=output:3
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=2, vlan_vid=0x1002 actions=output:2
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0, priority=2, in_port=1
actions=drop
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0, priority=2, in_port=2
actions=drop
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0, priority=2, in_port=3
actions=drop
```

Figura 103. Reglas de flujo de simple_switch_13_vlan.py

- Reglas para permitir tráfico en VLAN 1:

```
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=1, vlan_vid=0x1001 actions=output:3
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=3, vlan_vid=0x1001 actions=output:1
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=1, vlan_vid=0x1001 actions=output:1
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=3, vlan_vid=0x1001 actions=output:3
```

Figura 104. Reglas VLAN 1

- Reglas para permitir tráfico en VLAN 2:

```
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=1, in_port=2, vlan_vid=0x1002 actions=output:2
```

Figura 105. Reglas VLAN 2

Las reglas de flujo configuradas para permitir tráfico dentro de las VLAN son bastante específicas y tienen un propósito claro. Para **VLAN 1**, las reglas establecen que cualquier paquete que ingrese por el **puerto 1** (correspondiente a **enp0s8**) o el **puerto 3** (correspondiente a **enp0s10**) con el identificador de VLAN 1 será reenviado al puerto correspondiente dentro de la misma VLAN. Por ejemplo, un paquete que entra por el **puerto 1** se envía al **puerto 3** y viceversa. Estas reglas tienen una **prioridad de 1**, lo que significa que se aplicarán siempre que no haya reglas más específicas. De manera similar, para VLAN 2, cualquier paquete que ingrese por el **puerto 2** (correspondiente a **enp0s9**) con el identificador de **VLAN 2** se enviará de vuelta al mismo puerto, asegurando que el tráfico se mantenga dentro de la misma VLAN. La configuración asegura que el tráfico se mantenga segregado dentro de cada VLAN y facilita el manejo del tráfico de red de manera eficiente y segura.

- **Reglas Table-Miss:**

```
simple_switch_13_vlan.py
cookie=0x0, duration=0.199s, table=0, n_packets=0, n_bytes=0, priority=0
actions=CONTROLLER:65535
```

Figura 106. Regla Table Miss

Esta regla actúa como un salvavidas para cualquier tráfico que no coincida con ninguna de las otras reglas definidas. Con la prioridad más baja (0), esta regla captura cualquier paquete que no encuentre coincidencia en las reglas anteriores y lo envía al controlador para su procesamiento. Esto asegura que ningún paquete se pierda sin ser analizado, permitiendo al controlador decidir cómo manejar estos paquetes especiales. El buffer de 65535 bytes indica la cantidad máxima de datos que se enviarán al controlador por cada paquete coincidente.

- **Bloqueo de Tráfico entre VLANs:**

```
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=2,in_port=1 actions=drop
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=2,in_port=2 actions=drop
cookie=0x0, duration=10.199s, table=0, n_packets=0, n_bytes=0,
priority=2,in_port=3 actions=drop
```

Figura 107. Reglas de Bloqueo

Estas reglas están configuradas para bloquear cualquier dato que intente entrar por los puertos **1**, **2** y **3** correspondientes a **enp0s8**, **enp0s9** y **enp0s10** respectivamente. Con una prioridad de 2, estas reglas tienen suficiente importancia para asegurarse de que todos los paquetes que lleguen a estos puertos sean eliminados (drop). Esto es muy útil para prevenir que datos no deseados o no autorizados pasen por estos puertos específicos.

Con todo lo anterior y generando tráfico entre todos los dispositivos, se puede observar en la siguiente imagen que el tráfico está siendo correctamente bloqueado o enrutado siguiendo las reglas establecidas en el script:

```
loading app ryu.controller.ofp_handler
instantiating app vlan_switch_final.py of VLANIsolation
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK VLANIsolation
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'VLANIsolation': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7fbf37d52490> address:('127.0.0.1', 41228)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7fbf37dae730>
move onto config mode
EVENT ofp_event->VLANIsolation EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x660a0f77,OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796760147762,n_buffers=0,n_tables=254)
Bloqueando tráfico entre VLANs: in_port=1, out_port=2
Bloqueando tráfico entre VLANs: in_port=2, out_port=1
Bloqueando tráfico entre VLANs: in_port=3, out_port=2
Bloqueando tráfico entre VLANs: in_port=2, out_port=3
move onto main mode
```

Figura 108. Controlador Ryu bloqueando tráfico entre VLANs

Para verificar que las configuraciones de este apartado están correctamente implementadas, es necesario capturar el tráfico utilizando herramientas como tcpdump o Wireshark. Al realizar esta captura, podremos observar las etiquetas correspondientes a las VLAN 1 y 2:

- Tráfico entre dispositivos de una misma VLAN (VLAN 1):

```
▼ Frame 44: 46 bytes on wire (368 bits), 46 bytes captured (368 bits)
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun 21, 2024 19:28:28.681289000 Hora de verano romance
  UTC Arrival Time: Jun 21, 2024 17:28:28.681289000 UTC
  Epoch Arrival Time: 1718990908.681289000
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 0.000252000 seconds]
  [Time delta from previous displayed frame: 0.000252000 seconds]
  [Time since reference or first frame: 0.011524000 seconds]
  Frame Number: 44
  Frame Length: 46 bytes (368 bits)
  Capture Length: 46 bytes (368 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:vlan:ethertype:ip:icmp]
  [Coloring Rule Name: ICMP]
  [Coloring Rule String: icmp || icmpv6]
  ▼ Ethernet II, Src: RaspberryPi_29:ba:54 (2c:cf:67:29:ba:54), Dst: RaspberryPiF_54:e5:29 (b8:27:eb:54:e5:29)
    ▶ Destination: RaspberryPiF_54:e5:29 (b8:27:eb:54:e5:29)
    ▶ Source: RaspberryPi_29:ba:54 (2c:cf:67:29:ba:54)
    Type: 802.1Q Virtual LAN (0x8100)
  ▼ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 1
    000. .... .. = Priority: Best Effort (default) (0)
    ...0 .... .. = DEI: Ineligible
    ... 0000 0000 0001 = ID: 1
    Type: IPv4 (0x0800)
  ▶ Internet Protocol Version 4, Src: 10.0.0.5, Dst: 10.0.0.3
  ▶ Internet Control Message Protocol
```

Figura 109. Tráfico entre dispositivos de una misma VLAN

Capítulo 6. Conclusiones y trabajos futuros

En este proyecto, se han cumplido los objetivos planteados, que eran analizar el funcionamiento de una red SDN utilizando Open vSwitch y el controlador Ryu, además de emplear Raspberry Pi en las simulaciones. A lo largo del trabajo, se logró instalar y configurar Open vSwitch, implementar y utilizar el controlador SDN Ryu, y llevar a cabo simulaciones prácticas con Raspberry Pi. Estas actividades permitieron evaluar de manera práctica el rendimiento, la flexibilidad y la escalabilidad de las redes definidas por software en un entorno controlado y realista. A continuación, se resumen los hallazgos principales y se proponen posibles líneas de investigación para seguir explorando las ventajas y desafíos de las redes definidas por software.

A lo largo del desarrollo de este proyecto, hemos podido comprobar de primera mano las ventajas que ofrece una red definida por software (SDN) frente a las redes tradicionales. Utilizando Open vSwitch, el controlador Ryu y las Raspberry Pi, hemos llevado a cabo varios casos prácticos que nos han permitido sacar conclusiones claras:

- **Control Centralizado y Visibilidad de la Red:** Gracias al controlador Ryu, hemos podido gestionar y monitorizar toda la red desde un único punto. Esto nos ha dado una visión completa y precisa del estado de la red, algo que contrasta con la fragmentación y la dificultad de coordinación de las redes tradicionales.
- **Flexibilidad y Escalabilidad:** La configuración dinámica y automatizada de las SDN ha demostrado ser extremadamente adaptable a cambios en la topología de la red y en las necesidades de tráfico. Hemos podido implementar políticas de Calidad de Servicio (QoS) y VLANs de manera rápida y eficiente, sin necesidad de intervención manual constante, lo que subraya la flexibilidad de las SDN.
- **Reducción de Costos:** Al utilizar hardware genérico, como las Raspberry Pi y los switches virtuales, hemos conseguido reducir significativamente los costos de implementación y mantenimiento. Esto supone una gran ventaja frente a las redes tradicionales, que requieren hardware propietario más caro y menos flexible.
- **Optimización de Recursos y Mejor Rendimiento:** La capacidad de tomar decisiones de enrutamiento y conmutación basadas en una visión global de la red ha permitido una utilización más eficiente de los recursos. Esto se traduce en una mejor gestión de la congestión y en una reducción de la latencia y la pérdida de paquetes, tal y como hemos comprobado en nuestras pruebas.

Sin embargo, es importante mencionar que las SDN también presentan ciertos desafíos. La centralización del control puede suponer un punto único de fallo y aumentar la superficie de ataque, lo que requiere una atención constante a la seguridad y la redundancia del controlador. Además, la implementación inicial de las SDN puede requerir conocimientos técnicos especializados y una integración cuidadosa con las infraestructuras existentes.

Para el futuro, se sugiere investigar y desarrollar mecanismos robustos de seguridad y redundancia para el controlador Ryu, abordando la vulnerabilidad del control centralizado. También sería beneficioso ampliar el uso de herramientas de automatización y orquestación para gestionar redes SDN más grandes de manera eficiente, desarrollando scripts y algoritmos que optimicen la asignación de recursos y el balanceo de carga. Además, explorar cómo las SDN pueden integrarse con tecnologías emergentes como 5G, IoT y computación en la nube permitirá adaptar y escalar la infraestructura de red para soportar nuevas aplicaciones y servicios que requieren alta capacidad y baja latencia. Realizar pruebas exhaustivas de escalabilidad y rendimiento en entornos más grandes y complejos ayudará a identificar y resolver posibles cuellos de botella y problemas de rendimiento en despliegues a gran escala. Finalmente, desarrollar y probar aplicaciones específicas que aprovechen las capacidades de las SDN, como la gestión avanzada de QoS para aplicaciones en tiempo real y la implementación de políticas de seguridad dinámicas basadas en el comportamiento del tráfico, contribuirá a maximizar los beneficios de esta tecnología.



En resumen, las SDN representan un avance significativo en la gestión y operación de redes, ofreciendo beneficios claros en términos de flexibilidad, escalabilidad y costos. Sin embargo, es crucial seguir investigando y desarrollando soluciones para abordar los desafíos actuales y futuros, garantizando así una evolución continua y sostenible de la infraestructura de red. Este proyecto ha sido una valiosa experiencia práctica que ha confirmado las ventajas de las SDN y ha planteado interesantes oportunidades para futuros trabajos en esta área.

Capítulo 7. Bibliografía

- [1] S. Sinha, «State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally», *IoT Analytics*, 26 de enero de 2024. <https://iot-analytics.com/number-connected-iot-devices/>
- [2] J. Espinoza, «Serie SDN — El Controlador SDN y los Planos de Control», *Medium*, 30 de marzo de 2022. [En línea]. Disponible en: <https://jesuseduardoespinoza.medium.com/serie-sdn-el-controlador-sdn-y-los-planos-de-control-6b6f6a8e9bf1>
- [3] EMA, «Next-Generation Wide Area Networks», Infovista. Accedido: 5 de mayo de 2024. [En línea]. Disponible en: https://www.infovista.com/sites/default/files/resources/as_ema_nextgen_wan.pdf
- [4] Internet Society, «Una breve historia de Internet - Internet Society», *Internet Society*, 11 de octubre de 2023. <https://www.internetsociety.org/es/internet/history-internet/brief-history-internet/>
- [5] C. Informàtica, «WwW TIM BERNERS-LEE: biografía y legado » CIS Informàtica», *CIS Informàtica*, 14 de febrero de 2024. <https://www.cisinformatica.cat/es/tim-berners-lee-que-invento/>
- [6] «Understand Open Shortest Path First (OSPF) - Design Guide», *Cisco*, 8 de marzo de 2024. <https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html>
- [7] «IP Routing: BGP Configuration Guide - Cisco BGP Overview [Cisco ASR 1000 Series Aggregation Services Routers]», *Cisco*, 13 de septiembre de 2019. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xr-16/irg-xr-16-book/cisco-bgp-overview.html
- [8] Admin, «¿Cómo se calcula la métrica del protocolo OSPF?», *TutoManiac*, 13 de mayo de 2022. <https://tutomaniac.com/como-se-calcula-la-metrica-del-protocolo-ospf/>
- [9] M. B. Gil, «Análisis de las Ventajas y Desventajas de BGP: Todo lo que necesitas saber», *Ventajas y Desventajas Top*, 10 de noviembre de 2023. <https://ventajasydesventajastop.com/bgp-ventajas-y-desventajas/>
- [10] «RFC 7727: Spanning Tree Protocol (STP) Application of the Inter-Chassis Communication Protocol (ICCP)», *IETF Datatracker*. <https://datatracker.ietf.org/doc/rfc7727/>
- [11] «RFC 2326: Real Time Streaming Protocol (RTSP)», *IETF Datatracker*. <https://datatracker.ietf.org/doc/html/rfc2326>
- [12] M. Escalante, «Qué es el Spanning Tree Protocol (STP)», *abcXperts*, 17 de junio de 2023. <https://abcxperts.com/que-es-el-spanning-tree-protocol-stp/>
- [13] «Virtual Tunneling Protocol BOF (vtp)». <https://datatracker.ietf.org/wg/vtp/about/>
- [14] «RFC 7424: Mechanisms for Optimizing Link Aggregation Group (LAG) and Equal-Cost Multipath (ECMP) Component Link Utilization in Networks», *IETF Datatracker*. <https://datatracker.ietf.org/doc/html/rfc7424>
- [15] B. Lutkevich, «real-time application (RTA)», *Unified Communications*, 9 de mayo de 2022. <https://www.techtarget.com/searchunifiedcommunications/definition/real-time-application-RTA>
- [16] «What are the 3 Requirements of the Internet of Things (IoT)?» <https://www.tutorialspoint.com/what-are-the-3-requirements-of-the-internet-of-things-iot>

- [17] «5G technology and networks (speed, use cases, rollout)», *Thales Group*, 24 de febrero de 2023. <https://www.thalesgroup.com/en/markets/digital-identity-and-security/mobile/inspired/5G>
- [18] «¿Qué es cloud computing? | Google Cloud», *Google Cloud*. <https://cloud.google.com/learn/what-is-cloud-computing?hl=es>
- [19] P. Tikait, «Features Of Big Data Analytics | Top Big Data Requirements», 15 de abril de 2024. <https://www.selecthub.com/big-data-analytics/big-data-analytics-requirements/>
- [20] PricewaterhouseCoopers, «Perspectivas del panorama global de las telecomunicaciones 2023-2027», *PwC*, 10 de octubre de 2023. <https://www.pwc.com/co/es/pwc-insights/perspectivas-panorama-global-telecomunicaciones.html>
- [21] «A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks», *IEEE Journals & Magazine | IEEE Xplore*. <https://ieeexplore.ieee.org/document/6739370>
- [22] Roger, «Redes Definidas por Software (SDN): Tipos, Ventajas y Aplicaciones | Comunidad FS», *Knowledge*. <https://community.fs.com/es/article/software-defined-networking-sdn-types-advantages-and-applications.html>
- [23] M. Sepúlveda, «Interfaces de programación de aplicaciones (API) para el CCNA - eClassVirtual - Cursos Cisco en línea», *eClassVirtual - Cursos Cisco en línea*, 18 de septiembre de 2022. <https://eclassvirtual.com/interfaces-de-programacion-de-aplicaciones-api-para-el-ccna/>
- [24] «Welcome to RYU the Network Operating System(NOS) — Ryu 4.34 documentation». <https://ryu.readthedocs.io/en/latest/>
- [25] «noxrepo - Overview», *GitHub*. <https://github.com/noxrepo/>
- [26] «OpenDaylight». <https://www.opendaylight.org/>
- [27] «Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions», *Open Networking Foundation*, 11 de julio de 2022. <https://opennetworking.org/onos/>
- [28] «Home - Tungsten Fabric», *Tungsten Fabric*, 28 de julio de 2023. <https://tungsten.io/>
- [29] «Open vSwitch». <https://www.openvswitch.org/>
- [30] «Reinvent your data center with Cisco Nexus 9000 Series switches», *Cisco*, 8 de noviembre de 2023. https://www.cisco.com/c/m/en_us/products/switches/nexus-9000/nexus-9000-migration.html#/
- [31] Open Networking Foundation, *OpenFlow Switch Specification Version 1.3.0*. Open Networking Foundation, 2012. [PDF]. Disponible en: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>
- [32] «NETCONF User Guide — NetConf master documentation». <https://docs.opendaylight.org/projects/netconf/en/latest/user-guide.html>
- [33] «Ryu SDN Framework—Open-source SDN Platform Software | NTT Technical Review». <https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.html>
- [34] «Ryu SDN Framework». <https://ryu-sdn.org/index.html>
- [35] Ohsaki Kenichi, M. Nohara, M. Yoshimoto, Y. Kadobayashi, y K. Ogawa, *Ryu SDN Framework*. [En línea]. Disponible en: <https://book.ryu-sdn.org/en/Ryubook.pdf>
- [36] «VMware a la compra de Nicira y sus soluciones de plataforma de virtual networking», *Wetcom*, 9 de julio de 2022. <https://www.wetcom.com/blog/blog-1/vmware-a-la-compra-de-nicira-y-sus-soluciones-de-plataforma-de-virtual-networking-359>

- [37] «OpenFlow, democratizando SDN», *Entre Dev y Ops*, 18 de febrero de 2015. <https://www.entredevyops.es/posts/openflow-sdn.html>
- [38] R. P. Ltd, «Raspberry Pi», *Raspberry Pi*. <https://www.raspberrypi.com/>
- [39] GeeksforGeeks, «Difference between Software Defined Network and Traditional Network», *GeeksforGeeks*, 11 de mayo de 2023. <https://www.geeksforgeeks.org/difference-between-software-defined-network-and-traditional-network/>
- [40] SDxCentral, «How 5G SDN Will Bolster Wireless Networks - SDxCentral», *SDxCentral*, 4 de mayo de 2022. <https://www.sdxcentral.com/5g/definitions/key-elements-5g-network/5g-sdn/>
- [41] M. A. Al-Shareeda, A. A. Alsadhan, H. H. Qasim, y S. Manickam, «Software defined networking for internet of things: review, techniques, challenges, and future directions», *Bulletin Of Electrical Engineering And Informatics*, vol. 13, n.o 1, pp. 638-647, feb. 2024, doi: 10.11591/eei.v13i1.6386.
- [42] V. R. César *et al.*, «Security automation in software defined networks», 1 de junio de 2023. <https://repositorio.tec.mx/handle/11285/651165>
- [43] D. Linthicum, «How to use SDN and cloud to manage your network infrastructure», *TechBeacon*, 22 de enero de 2019. <https://techbeacon.com/enterprise-it/pair-sdn-cloud-more-efficient-network-management>
- [44] «Download Ubuntu Desktop | Ubuntu», *Ubuntu*. <https://ubuntu.com/download/desktop>
- [45] «Oracle VM VirtualBox». <https://www.virtualbox.org/>
- [46] Faucetsdn, «GitHub - faucetsdn/ryu: Ryu component-based software defined networking framework», *GitHub*. <https://github.com/faucetsdn/ryu>
- [47] Pyenv, «GitHub - pyenv/pyenv: Simple Python version management», *GitHub*. <https://github.com/pyenv/pyenv>
- [48] «pip», *PyPI*, 3 de febrero de 2024. <https://pypi.org/project/pip/>
- [49] «get-pip.py», *Bootstrap.pypa.io*. <https://bootstrap.pypa.io/get-pip.py> (accedido 2 de junio de 2024).
- [50] «Qué es ICMP (Protocolo de control de mensajes de Internet) | Fortinet», *Fortinet*. <https://www.fortinet.com/lat/resources/cyberglossary/internet-control-message-protocol-icmp>
- [51] Amirashoori, «sdn_qos/ryu_qos_apps/qos_simple_switch_13.py at main · amirashoori7/sdn_qos», *GitHub*. https://github.com/amirashoori7/sdn_qos/blob/main/ryu_qos_apps/qos_simple_switch_13.py
- [52] «Quality of Service (QoS) — Open vSwitch 3.3.90 documentation». <https://docs.openvswitch.org/en/latest/faq/qos/>
- [53] Esnet, «GitHub - esnet/iperf: iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool», *GitHub*. <https://github.com/esnet/iperf>
- [54] «VLANs—Open vSwitch 3.3.90 documentation». <https://docs.openvswitch.org/en/latest/faq/vlan/>
- [55] S. De Luz, «VLANs: Qué son, tipos y para qué sirven», *RedesZone*, 29 de mayo de 2024. [En línea]. Disponible en: <https://www.redeszone.net/tutoriales/redes-cable/vlan-tipos-configuracion/>
- [56] ManageEngine, «Puerto de acceso vs. Puerto troncal | Access port vs. Trunk port - ManageEngine OpUtils». <https://www.manageengine.com/latam/oputils/tech-topics/puerto-de-acceso-vs-puerto-troncal.html>



- [57] «Basic Configuration — Open vSwitch 3.3.1 documentation».
<https://docs.openvswitch.org/en/stable/faq/configuration/>
- [58] Faucetsdn, «ryu/ryu/app/simple_switch_13.py at master · faucetsdn/ryu», *GitHub*.
https://github.com/faucetsdn/ryu/blob/master/ryu/app/simple_switch_13.py
- [59] «Putting hosts in vlan using Ryu controller», *Stack Overflow*.
<https://stackoverflow.com/questions/72810261/putting-hosts-in-vlan-using-ryu-controller>