UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Carrera de
Electrónica y Telecomunicaciones

Diseño e implementación de un entorno de análisis, experimentación, evaluación y desarrollo de Redes Definidas por Software (SDN)

Trabajo de titulación previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones.

Autores :

Jonnathan Fernando Nivicela Arbito — C.I. 010542559-9
jfer-junior@hotmail.com

Jhoselin Adriana Vera Peláez — C.I. 140079778-1
jhoss_22.9_7@hotmail.com

Director :

Ing. Andrés Marcelo Vázquez Rodas, PhD — C.I. 030149684-0

Cuenca - Ecuador

28 de septiembre de 2021

Resumen

Las redes definidas por software (Software-Defined Networking (SDN)), permiten el desacoplamiento del plano de datos con respecto al plano de control, facilitando tanto a los desarrolladores de aplicaciones como a los administradores el control uniforme y centralizado sobre los elementos de red. El controlador es aquel que posee visión total de la red e intercambia mensajes con los conmutadores indicando cual será el comportamiento de la red en base a diversos criterios.

Las SDN representan un paradigma emergente de redes modernas que no puede pasarse por alto, por lo que varias universidades tanto nacional como internacionalmente han desarrollado trabajos basados en SDN, por tanto, se considera importante y trascendental que la Universidad de Cuenca aborde la investigación de este tema; partiendo del manejo de escenarios, configuración y desarrollo de aplicaciones SDN.

En este trabajo se plantea el diseño e implementación de un entorno de análisis, experimentación, evaluación y desarrollo de SDN, este entorno puede ser utilizado como punto partida para el estudio y despliegue de SDN en la Universidad de Cuenca, es así que este Trabajo de Titulación es pionero en abordar este tema dentro de la Alma mater.

El entorno de pruebas se compone de 12 prácticas, cada una cuenta con un escenario diferente, donde se ejemplifican y evalúan aplicaciones existentes y se desarrollan variantes con la finalidad de dar un paso adelante cubriendo desafíos y proponiendo soluciones para las redes modernas. Los principales temas a abordar en el desarrollo de estas prácticas son: Ingeniería de tráfico (Traffic Engineering (TE)), monitoreo, administración y seguridad, calidad de servicio (Quality of Service (QoS)), entre otras. Dichas prácticas no solo se limitan a un entorno de emulación sino que también se llevan a un entorno de implementación física, es así que algunas de ellas, las más didácticas para el aprendizaje de SDN, son implementadas en un laboratorio de pruebas conformado por hardware asequible (Raspberry Pi Model 3B y Rapsberry Pi Model 4) para los conmutadores de red y ordenadores portátil como terminales y controladores; siendo necesario el uso adicional de adaptadores bus universal en serie (Universal Serial Bus (USB))-Ethernet.

En base a lo realizado se han elaborado las guías de prácticas correspondientes, que puedan ser usadas como material didáctico y de apoyo para estudio, investigación y desarrollo de redes definidas por software en la Universidad de Cuenca.

Palabras clave: SDN. Redes definidas por software. Plano de datos. Plano de control. Raspberry Pi. Controladores SDN. Emulación. Ryu. POX. OpenDayLight. Open vSwitch. Mininet. OpenFlow. Guías Prácticas. Banco de Pruebas.

Abstract

Software Defined Networks (SDN), allow the decoupling of the data plane regarding the control plane, facilitating both application developers and administrators the uniform and centralized control over the network elements. It is the controller who has a total vision of the network and exchanges messages with the switches indicating what the behavior of the network will be based on various criteria.

The SDN represent an emerging paradigm of modern networks that cannot be ignored, which is why several universities both nationally and internationally have developed papers based on SDN, which is why it is considered important and transcendental that the University of Cuenca address research on this topic; starting from the handling of scenarios, configuration and application development SDN.

This work propound the design and implementation of an environment for analysis, experimentation, evaluation and development of Software Defined Networks (SDN), this environment can be used as a starting point for the study and deployment of SDN at the University of Cuenca, turning this academic work into a forerunner paper to adress this issue within the Alma mater.

The try-out environment is made up of 12 practices, each one has a different scenario, where existing applications are exemplified and evaluated and variants are developed in order to take a step forward, covering challenges and proposing solutions for modern networks. The main topics to be addressed in the development of these practices are: Traffic Engineering, Monitoring, Administration and Security, QoS, among others. These practices are not only limited to an emulation environment but are also taken to a physical implementation environment, thus the most didactic ones for learning SDN, are implemented in a test laboratory with affordable hardware (Raspberry Pi Model 3B and Rapsberry Pi Model 4) for network switches and laptops such as terminals and controllers, requiring additional USB-Ethernet adapters.

Based on the work carried out, the corresponding practice guides will be prepared so that they can be used as didactic and support material for the study, research and development of Software Defined Networks at the University of Cuenca.

Keywords: SDN. Software-Defined Networking. Data plane. Control plane. Raspberry Pi. SDN controllers. Emulation. Ryu. POX. OpenDayLight. Open vSwitch. Mininet. OpenFlow. Practical Guides. Testbed.

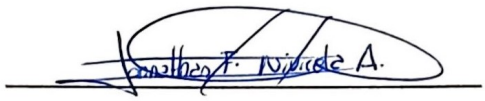
Índice general

Índice de figuras

Cláusula de Propiedad Intelectual

Jonnathan Fernando Nivicela Arbito, autor del trabajo de titulación "Diseño e implementación de un entorno de análisis, experimentación, evaluación y desarrollo de Redes Definidas por Software (SDN)", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 28 de septiembre de 2021



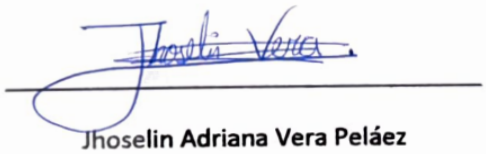
Jonnathan Fernando Nivicela Arbito

C.I: 0105425599

Cláusula de Propiedad Intelectual

Jhoselin Adriana Vera Peláez, autora del trabajo de titulación "Diseño e implementación de un entorno de análisis, experimentación, evaluación y desarrollo de Redes Definidas por Software (SDN)", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autora.

Cuenca, 28 de septiembre de 2021



Jhoselin Adriana Vera Peláez

C.I: 1400797781

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Jonnathan Fernando Nivicela Arbito en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Diseño e implementación de un entorno de análisis, experimentación, evaluación y desarrollo de Redes Definidas por Software (SDN)", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 28 de septiembre de 2021

Jonnathan Fernando Nivicela Arbito

C.I: 0105425599

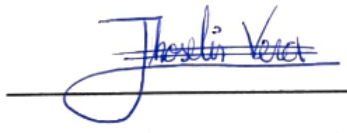
UNIVERSIDAD DE CUENCA

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Jhoselin Adriana Vera Peláez en calidad de autora y titular de los derechos morales y patrimoniales del trabajo de titulación "Diseño e implementación de un entorno de análisis, experimentación, evaluación y desarrollo de Redes Definidas por Software (SDN)", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 28 de septiembre de 2021



Jhoselin Adriana Vera Peláez

C.I: 1400797781

Dedicatoria

A mis padres, Clemencia y Alfonso, quienes desde mis primeros pasos han sido mi guía y apoyo incondicional para poder alcanzar cada peldaño en mis estudios, les debo mi formación como profesional así como mi crecimiento como persona.

A mis abuelos, especialmente a **mami Blanca** por ser una segunda madre fomentándome valores y principios; así como el amor por el estudio y la superación; por ser la persona que día a día espera con ansias que alcance mis metas académicas. Así mismo, a mi abuelo **Cornelio** por su carácter, que en mucho, me ha formado como una persona fuerte y luchadora. A mis abuelos paternos, abuelo **José**, aunque ausente en este mundo pero su legado y apellido perduran en mí, así también a mi abuela **Mercedes**.

A mis hermanos maternos: Lucrecia, Luis Alberto, Oscar David y Rosa Guadalupe; y a mis **hermanos paternos, Alexander y David**, por ser un apoyo emocional para sonreír cuando los días son grises, por ser uno de mis motivos para superarme y dejar un buen ejemplo académico.

A mi tía Julia y a mis primos, por convertirse en mi segunda familia y compañía desde mi temprana edad, cuando llegué a la ciudad a seguir el sueño de superarme; por su presencia y apoyo en la medida de lo posible. Especialmente a Ximena y Marcia, quienes han sido un ejemplo académico de superación; también a Jessenia por su amistad y apoyo anímico.

A mis amigos por su apoyo moral, en especial a aquellas personas que anhelaron estar presentes en este instante de mi vida académica y que, por diversas situaciones, ya no están presentes en mi vida; por haber sido motivación, aliento y compañía en cierto momento de este largo camino.

Jonnathan F. Nivicela Arbito

Dedicatoria

A mis padres José y Rosa:

Por el apoyo incondicional brindado, no solamente a lo largo de mi carrera universitaria sino a lo largo de la vida, por su amor y comprensión a pesar de la distancia y adversidades. Por siempre preocuparse por mi futuro e incentivarme a superarme. Porque gracias a ellos he podido superar obstáculos e ir cumpliendo cada meta planteada.

A mis abuelos José y Virginia:

Por brindarme su apoyo, por ser mi aliento y compañía en los momentos que más los necesite. Porque ellos, con su ejemplo de perseverancia, me enseñaron a salir adelante, luchar por mis sueños y no darme por vencida. Porque bastaba un abrazo de ellos para saber que todo saldría bien, por encontrar las palabras precisas para animarme y estar siempre pendientes de mi y de mis hermanos.

A mis hermanos Tania, John y Jeison:

Por estar siempre presentes y darme su apoyo cada vez que lo necesitaba. Por darme ánimos y enseñarme siempre a ver el lado bueno de las cosas, por ser un equipo para salir adelante como familia.

A Esteban:

Por su amor, compresión, compañía, apoyo y paciencia en cada momento. Por darme ánimo y levantarme en los momentos difíciles, por enseñarme a creer en mi y recordarme que soy capaz de superar cada obstáculo que se me presente. Por ser un pilar fundamental en mi vida y una motivación para salir adelante.

A mis amigos y compañeros, por apoyarme y acompañarme a lo largo de la carrera, por la risas, amanecidas y enseñanzas compartidas. Por haber estado presentes de una u otra manera.

Jhoselin A. Vera Peláez

Agradecimientos

A nuestras familias por el apoyo incondicional y la confianza en nosotros durante todo este tiempo.

Queremos expresar un agradecimiento especial a nuestro director, Ing. Andrés Vázquez; quien, con sus conocimientos y experiencia nos ha guiado y apoyado en las diferentes etapas de este Trabajo de Titulación; logrando, en equipo, alcanzar los objetivos planteados.

También extendemos este agradecimiento al Ing. Fabián Astudillo así como al Ing. Santiago González por su asesoría, tiempo y atención al presente Trabajo de Titulación; no hubiésemos alcanzado las metas propuestas sin su apoyo.

De igual manera, agradecemos a todos nuestros docentes universitarios por compartir sus conocimientos y experiencia, lo que nos ha permitido formarnos académicamente.

Finalmente agradecemos a nuestros amigos y compañeros por su apoyo en arduas jornadas de estudio y esfuerzo conjunto para alcanzar y superar cada reto académico que se nos ha presentado.

LOS AUTORES

Abreviaciones y Acrónimos

AAA Authentication, Authorization and Accounting. 57, 140, 141
ACK ACKNOWLEDGMENT. 104
API Application Programming Interfaces. 3, 13, 20, 21, 26, 57, 75, 132, 133, 140, 150
ARM Advanced RISC Machine. 31, 32
ARP Address Resolution Protocol. 19, 20, 25, 36, 120

BGP Border Gateway Protocol. 35, 36, 58, 147–153
BIRD BIRD Internet Routing Daemon. 147

CAPEX Capital Expenditures. 13
CEDIA Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia. 4
CISC Complex Instruction Set Computer. 32
CLI Command-Line Interface. 21, 27, 28, 77, 81, 85, 87, 102, 122, 123, 128–130, 134, 140, 150, 156
CPU Central Processing Unit. 31

DHCP Dynamic Host Configuration Protocol. 4, 8
DLUX OpenDayLigth User eXperience. 50, 52, 75
DNS Domain Name System. 4, 8, 20
DoS Denial of Service. 4, 8, 10
DPDK Data Plane Development Kit. 23
DPID Datapath Identifier. 16, 18

EPN Escuela Politécnica Nacional. 3, 4, 7, 8
ESPE Escuela Superior Politécnica del Ejercito. 3, 7, 8
ESPOCH Escuela Superior Politécnica de Chimborazo. 4, 7–9
ESPOL Escuela Politécnica del Litoral. 3, 7, 8

FTP File Transfer Protocol. 18, 56, 106, 110–112

GENEVE Generic Network Virtualization Encapsulation. 21
GNU GNU's Not Unix. 25, 26
GRE Generic Routing Encapsulation. 21, 154
GUI Graphical User Interface. 10, 24, 27, 30

HDMI High-Definition Multimedia Interface. 32, 38, 42
HTTP Hypertext Transfer Protocol. 4, 8, 56, 57, 106, 110, 112, 113, 132, 134, 138, 139, 150

ICMP Internet Control Message Protocol. 4, 8, 57, 101, 136, 138, 159

IDS/IPS Intrusion Detection System/Intrusion Prevention System. 4, 8, 9
IoT Internet of Things. 1
IP Internet Protocol. 3, 4, 8, 18, 43, 54, 66, 75, 77, 86, 99, 109, 111, 112, 125, 126, 129, 147
IPFIX Internet Protocol Flow Information Export. 21
IPv4 Internet Protocol version 4. 101, 108
IPv6 Internet Protocol version 6. 4, 8, 9

JAR Java Archive. 21

LACP Link Aggregation Control Protocol. 21, 53, 82, 83, 86–89
LAN Local Area Network. 43
LISP Locator/ID Separation Protocol. 21
LLDP Link Layer Discovery Protocol. 19, 93, 101, 107, 108, 123
LTS Long Term Support. 21

M2M Machine to Machine. 1
MAC Media Access Control. 15, 19, 20, 67, 76–78, 84, 86, 94, 99, 101, 108
Mac OS Macintosh Operating System. 20
MPLS Multiprotocol Label Switching. 4, 8, 125, 126, 154

NFV Network Functions Virtualization. 4, 7, 8, 26
NIC Network Interface Controller. 21, 22, 36
NIC Network Intent Composition. 36, 57, 140–142, 145
NS3 Network Simulator 3. 24, 25

ODL OpenDayLight. 4, 7–9, 19–21, 35, 36, 49, 50, 52, 53, 57, 62, 65–69, 72, 73, 75–77, 79–81, 140–142, 147–153
OPEX Operational Expenses. 13
OSGi Open Services Gateway initiative. 21
OSPF Open Shortest Path First. 147
OVS Open vSwitch. 3, 4, 8, 11, 21–23, 37, 58, 63

P4 Programming Protocol-independent Packet Processors. 25
PID Process ID. 26

QoE Quality of Experience. 2
QoS Quality of Service. 2, 4, 8, 10, 13, 21, 54, 56, 82

RADIUS Remote Access Dial In User Service. 9
RAM Random Access Memory. 32
REST Representational State Transfer. 3, 20, 21, 57, 132, 133, 150
RIB Routing Information Base. 15
RIP Routing Information Protocol. 147
RISC Reduced Instruction Set Computer. 31, 32
RPi Raspberry Pi. 3–5, 7–11, 31–34, 37, 38, 42–46, 48
RPM RPM Package Manager. 22
RSPAN Remote Switched Port Analyzer. 21

RTCP Real Time Transport Protocol. 56, 106, 110, 114–116
RTP Real Time Transport Protocol. 56, 106, 110, 113–115, 117
RTSP Real Time Streaming Protocol. 60, 61

SAL Service Abstraction Layer. 21
SD Secure Digital. 32, 38
SD-WAN Software-Defined Wide-Area Network. 4, 8
SD-WSN Software-Defined Wireless Sensor Network. 9
SDN Software-Defined Networking. 2–15, 19, 21, 24–26, 35–37, 50, 57–59, 61–63, 65, 66, 72, 75, 76, 83, 91, 92, 100, 106, 107, 119, 121, 125, 126, 132, 133, 141, 147, 148, 150, 154, 155
sFlow sampled Flow. 21
SMTP Simple Mail Transfer Protocol. 18
SNMP Simple Network Management Protocol. 18
SSH Secure Shell. 32, 42, 46
STP Spanning Tree Protocol. 30
STT Stateless Transport Tunneling. 21

TCP Transmission Control Protocol. 18, 36, 38, 55–58, 60, 61, 68, 78, 99, 100, 102–105, 108–112, 134, 137, 138, 147
TCP/IP Transmission Control Protocol/Internet Protocol. 1
TE Traffic Engineering. 3, 5, 6, 10, 13, 35, 57, 62

UCE Universidad Central del Ecuador. 7
UDP User Datagram Protocol. 18, 36, 55, 56, 58–61, 68, 97, 99, 100, 102–106, 108–110, 113, 114, 117
UG Universidad de Guayaquil. 4, 7, 8
UI User Interface. 25
UNAN Universidad Nacional Autónoma de Nicaragua. 9
UniCan Universidad de Cantabria. 10
UNL Universidad Nacional de Loja. 7, 8
UPS Universidad Politécnica Salesiana. 4, 7–9
UPV Universidad Politécnica de Valencia. 10
USB Universal Serial Bus. 5, 32, 37, 39, 40, 47, 58
UTA Universidad Técnica de Ambato. 4

VLAN Virtual Local Area Network. 10, 21, 99
VM Virtual Machine. 21, 22, 25, 154
VXLAN Virtual eXtensible Local Area Network. 4, 8, 21, 35, 36, 58, 154–156, 158–160

WiFi Wireless Fidelity. 32, 38, 43, 59
WLAN Wireless Local Area Network. 9
WSN Wireless Sensor Network. 9

CAPÍTULO 1

Introducción

Este capítulo presenta el contexto evolutivo de la arquitectura de red tradicional de Internet hasta la actualidad remarcando sus ventajas y limitaciones, mismas que han obligado a abordar nuevos paradigmas para el próximo paso en dicha evolución. En la primera parte se describe la problemática ocasionada por diferentes factores, que ha motivado el surgimiento de un nuevo enfoque y paradigma de arquitectura del Internet (Sección 1.1). A continuación, se expone una contribución al nuevo enfoque que emerge como solución de la problemática planteada y se establece la dirección del presente trabajo (Sección 1.2). Luego, se define el alcance de este proyecto (Sección 1.3) y se plantean los objetivos (Sección 1.4). Finalmente, el capítulo presenta una visión general de la estructura del presente documento (Sección 1.5).

1.1. Identificación del problema

Desde el origen del Internet, hace más de 5 décadas, la red tradicional se ha adaptado convenientemente a las demandas de tráfico y aplicaciones bajo la arquitectura y paradigma mayoritariamente cliente/servidor, con enlaces y topologías facilitadas por la presencia y constante evolución de los enrutadores. La arquitectura original del Internet, con su paradigma de conmutación de paquetes y control de red distribuido, y sus protocolos más significativos de control de transmisión y protocolo de Internet, Transmission Control Protocol/Internet Protocol (TCP/IP), con mínimas actualizaciones ha soportado por más de 40 años una infinidad de nuevos servicios, aplicaciones y la interconexión de una diversidad de dispositivos finales de usuario. Es así que se ha iniciado una masificación de dispositivos conectados a Internet que incluye el Internet de las cosas (Internet of Things (IoT)) y las comunicaciones máquina a máquina (Machine to Machine (M2M)). Sin embargo, la explosión de la computación en la nube y en la niebla, las redes de distribución de contenidos, la masificación del IoT y las comunicaciones M2M a escala mundial, la diversidad de dispositivos finales, el *big data*, la demanda cada vez más creciente de tráfico móvil y tráfico multimedia, entre otros, han motivado una reevaluación de los enfoques y arquitecturas de red tradicionales.

Entre las limitaciones más relevantes del enfoque y arquitectura de las redes TCP/IP tradicionales,

motivadas por estas nuevas aplicaciones y usos del Internet, destacan principalmente:

- Una arquitectura de red de núcleo rígida, estática y cada vez más compleja.
- La inhabilidad de un escalamiento aún mayor que el actual.
- La dependencia que los operadores y proveedores de servicio tienen con los fabricantes de los equipos de red.
- Los requisitos crecientes de calidad de servicio (QoS) y calidad de experiencia (Quality of Experience (QoE)) no contemplados en el diseño del Internet original, al igual que los requisitos de seguridad a todo nivel.
- Un enrutamiento rígido basado únicamente en dirección de destino y con un control distribuido en los enrutadores de cientos de miles de sistemas autónomos.
- Las políticas inconsistentes entre los distintos sistemas autónomos, cada uno cuenta con sus propios protocolos y políticas de enrutamiento.
- Los enrutadores tienen implícita la lógica de control sobre la lógica de conmutación y envío de los paquetes, realizando un control totalmente distribuido.

Los nuevos servicios, usuarios y aplicaciones demandan, por tanto [3]:

- Redes adaptables que puedan ajustarse y responder dinámicamente a los requisitos de los usuarios, las aplicaciones y las condiciones de la red.
- La automatización de ciertas funciones y procesos como la aplicación de políticas sin depender de cambios de configuración individuales en cada elemento de red.
- Un manejo eficiente de la movilidad de los usuarios.
- Mecanismos de seguridad integrados y en todo nivel.
- La posibilidad de crecimiento y escalabilidad bajo demanda.
- Un manejo y administración de red fácil y eficiente, con software y hardware independiente del proveedor.
- Un enrutamiento más flexible que sea capaz de ajustarse fácilmente a los requisitos de calidad de servicio y calidad de experiencia demandados por los usuarios.
- La capacidad de programabilidad y virtualización de la red, explotando los avances en las tecnologías de virtualización y computación en la nube.

En este contexto, surgen las redes definidas por software (SDN). Estas buscan cumplir con la evolución de la red bajo los requisitos de adaptabilidad, automatización, mantenibilidad, gestión del modelo, movilidad, seguridad integrada y escalado bajo demanda [6]. La idea fundamental de las SDN es el desacoplamiento del plano de datos del plano de control. Es decir, los enrutadores del núcleo de la red se encargarán únicamente del reenvío de los paquetes (plano de datos), en tanto que toda la lógica de control de enrutamiento y demás se ejecutará en uno o más controladores de la SDN (plano de control). Entonces, las SDN permitirían la integración entre las aplicaciones y la infraestructura de red, facilitando a los desarrolladores y administradores ejercer un control uniforme sobre los elementos de la red. El enfoque SDN divide en dispositivos separados la función de conmutación, reenvío de paquetes, como plano de datos; y la administración, descubrimiento de rutas y políticas de prioridades, como plano de control. Es ahora el controlador quien tiene una visión general y total de la red mientras que los conmutadores enlazan los caminos definidos por el controlador bajo distintos criterios.

La arquitectura de las SDN consta de las interfaces Norte, Sur, Este/Oeste. La interfaz Norte es la encargada de conectar el plano de aplicación con el plano de control, permitiendo que desarrolladores de

aplicaciones puedan "programar" el comportamiento requerido de la red de acuerdo a las necesidades de los usuarios de dicha red. La transferencia de estado representacional (Representational State Transfer (REST)), que corresponde a un estilo estructural de comunicaciones usadas por las aplicaciones, se ha convertido en una forma estándar de construir una interfaz de programación de aplicaciones (Application Programming Interfaces (API)) para controladores SDN en dirección norte. La interfaz Sur permite la comunicación entre el controlador y el plano de datos. OpenFlow se ha constituido en la interfaz Sur por defecto para las primeras soluciones SDN existentes. Finalmente, las interfaces Este/Oeste permiten la comunicación entre distintos controladores de red, en caso de existir más de uno en un mismo sistema autónomo o de manera general, permite la comunicación de los controladores de distintos sistemas autónomos.

En este trabajo se pretende por tanto abordar este nuevo paradigma de redes definidas por software desde una perspectiva práctica, generando una guía de prácticas de laboratorio que permita servir de base a estudiantes y profesionales interesados en introducirse en esta nueva forma de abordar las redes de datos.

1.2. Justificación

Siendo las SDN un paradigma y tecnología emergente, la academia ecuatoriana, y en particular la Universidad de Cuenca, requiere abordar la investigación desde el manejo de escenarios, configuraciones, y desarrollo de aplicaciones para redes definidas por software. Son necesarios entornos de desarrollo en los que los estudiantes repliquen con facilidad, evalúen técnicamente el comportamiento y propongan soluciones a problemas de red específicos, partiendo desde una base sólida y preestablecida. Es aquí, que este trabajo propone el desarrollo de un entorno de análisis y experimentación del paradigma de redes definidas por software. Se pretende que este entorno sirva para estudiar las SDN, y para analizar las ventajas y puntos débiles de aplicaciones ya existentes; de tal forma que se aborde alguna variante partiendo de la experimentación para mejorar parámetros de rendimiento de las redes en general. Las aplicaciones a considerarse estarán principalmente relacionadas con las siguientes áreas prioritarias: Ingeniería de tráfico (TE), seguridad, medición, monitoreo y administración, redes de centro de datos, entre otras.

Trabajos similares se han desarrollado académicamente en otras universidades del país. Por ejemplo, en la Escuela Politécnica del Litoral (ESPOL) [7] se ha implementado un banco de pruebas de costo efectivo para la comparación del rendimiento de ancho de banda y tiempos de respuesta de una SDN frente a una red tradicional. Para esto, se ha utilizado el hardware de desarrollo RPi 3B como conmutador de red y un controlador SDN con Open vSwitch (OVS) y Floodlight, respectivamente.

Dentro de la Escuela Politécnica Nacional (EPN) [8] se desarrolló un prototipo de SDN basado en herramientas de código abierto, RPi (distintos modelos) como conmutadores y controlador, usando OVS y Ryu respectivamente. El objetivo del trabajo es comparar la tasa efectiva y las variaciones del retardo respecto a emulaciones basadas en el software Mininet. Se propone además la implementación de una aplicación de *Firewall* aplicando reglas de restricción a distintos protocolos de aplicación y al protocolo de Internet (Internet Protocol (IP)).

En la Escuela Superior Politécnica del Ejercito (ESPE) [9] se ha implantado un *testbed* para una red inalámbrica utilizando SDN. Aquí se profundiza en escenarios de emulación en Mininet y de manera física en un nodo de conexión con el enrutador Mikrotik que tienen soporte para OpenFlow, se usa

OVS y ODL.

En el centro del país, en la Escuela Superior Politécnica de Chimborazo (ESPOCH) primero se desarrolló una plataforma para evaluar QoS en SDN [10] mediante el emulador Mininet. Se utiliza Ryu como controlador por la facilidad del lenguaje de programación Python. En el trabajo se aborda el diseño de escenarios, pruebas de conectividad, captura de tráfico y mensajes con Wireshark, la evaluación del rendimiento ante el bloqueo y permiso del tráfico del protocolo de mensajes de control de Internet (ICMP). Posteriormente, se propuso la integración de un sistema de detección/prevención de intrusos (Intrusion Detection System/Intrusion Prevention System (IDS/IPS)) Snort al controlador SDN para protección ante ataques de denegación de servicio (Denial of Service (DoS)) en escenarios SDN [11]. Para esto se utilizó el conmutador Zodiac Fx, y nuevamente Ryu en el controlador. Además, se evalúa la interoperabilidad con servicios Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS) y HTTP.

En la Universidad de Guayaquil (UG) [12] se diseñó e implementó un prototipo de redes de área amplia definidas por software (Software-Defined Wide-Area Network (SD-WAN)) basado en RPi en dos nodos de borde enlazados mediante un conmutador OVS y empleando a ODL en el controlador. Se logró crear y comunicar un túnel mediante el protocolo de red de área local extensible virtual (VXLAN) que permitió evaluar los tiempos de latencia.

Por su parte, en la Universidad Politécnica Salesiana (UPS) se ha diseñado y desplegado virtualización de funciones de red (Network Functions Virtualization (NFV)) sobre SDN en infraestructura virtual aplicando balanceo de carga y seguridad distribuida para entornos Internet Protocol version 6 (IPv6) [13]. También se ha desarrollado una SDN aplicando conmutación de etiquetas multiprotocolo (Multiprotocol Label Switching (MPLS)) con políticas de QoS para telefonía IP empleando dispositivos Mikrotik [14]. En ambos casos, nuevamente, se usa ODL y OVS.

Por último, Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia (CEDIA) [15] trabaja en la implementación de un banco de prototipos de SDN en arquitecturas virtuales que luego se llevarán a una infraestructura física propia de CEDIA apoyada en conmutadores habilitados de bajo costo. Este trabajo incluye la participación de la EPN, UPS y Universidad Técnica de Ambato (UTA).

1.3. Alcance

Con los antecedentes de la Sección 1.2, el entorno SDN propuesto en este trabajo busca integrar características comunes de bancos de pruebas mencionados anteriormente y contribuir con soluciones de red no abordadas en dichos trabajos. Considerará topologías de red más completas y complejas que las previamente tratadas. Se utilizarán tecnologías asequibles como RPi y ordenadores que servirán como conmutadores y controladores. A nivel de software, sobre el controlador se evaluarán las opciones más relevantes como ODL, Ryu, POX. Mientras que OVS permitirá crear los nodos conmutadores sobre las RPi (con el sistema operativo Raspberry Pi OS) u ordenadores Linux. Es importante mencionar que ODL y OVS son quienes llevan la vanguardia en el plano de control y de datos, respectivamente. La topología propuesta en su mayor complejidad se presenta en la Figura 1.1. El objetivo es que la arquitectura sea escalable, partiendo desde el escenario más simple que cuenta con un solo conmutador (SW0), pasando luego a 2 conmutadores (SW0 y SW2) y por último a la topología final mostrada con al menos 3 conmutadores. En primera instancia, se busca establecer la comunicación entre hosts

de un mismo sistema autónomo. Posteriormente es posible establecer la interconexión incluso entre diferentes sistemas autónomos y servidores externos. La topología escalable se logra mediante el uso de adaptadores USB-Ethernet, lo que permite manejar varias interfaces Ethernet desde los conmutadores implementados en las RPi. Se ha considerado tal topología para lograr implementar las aplicaciones que se seleccionen entre Ingeniería de tráfico (TE), seguridad, medición o monitoreo y administración.

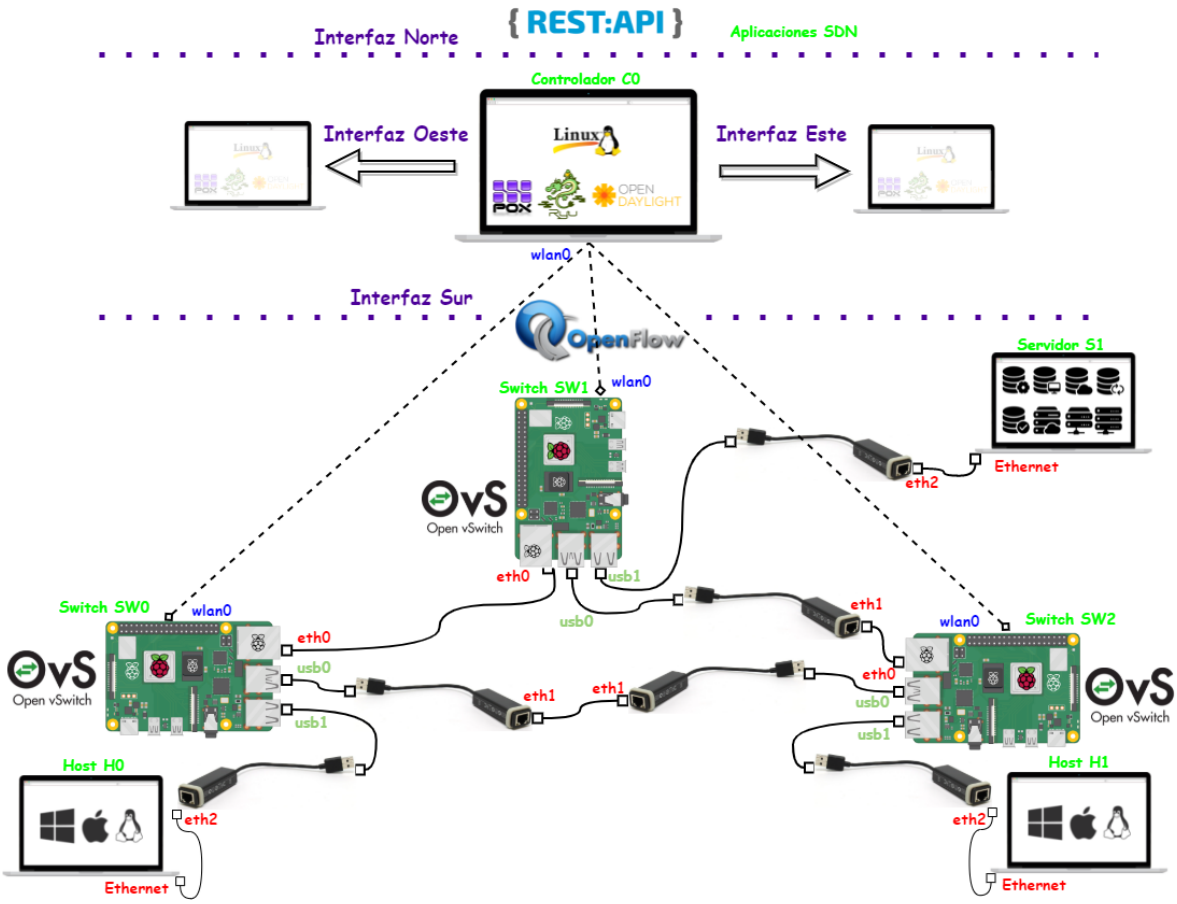


Figura 1.1: Topología del escenario de mayor complejidad.

Finalmente, tras plantear una topología SDN práctica se establecerán las configuraciones necesarias para la conectividad total de la red que luego faciliten las pruebas, selección e implementación de aplicaciones existentes. Una vez que sea viable la evaluación y observación del funcionamiento de la SDN y sus principales utilidades, se procederá a realizar una variante que se considere necesaria y aporte una nueva funcionalidad a la red. Todo el proceso y los resultados se llevarán con su debida guía y documentación que luego pueda servir como recursos académicos para prácticas, proyectos finales de ciclo, o trabajos de titulación; en general, motivando así la investigación y abordando el nuevo paradigma de SDN dentro de la Universidad de Cuenca.

1.4. Objetivos

1.4.1. Objetivo general

Diseñar e implementar un entorno de análisis, experimentación, evaluación y desarrollo de redes definidas por software (SDN), con su guía y documentación, que fundamente la propuesta de soluciones a los desafíos de las redes modernas.

1.4.2. Objetivos específicos

El presente trabajo tiene los siguientes objetivos específicos:

- Diseñar e implementar un entorno de pruebas para SDN, utilizando equipo tecnológico de costo asequible.
- Evaluar las distintas opciones de controladores para SDN y seleccionar la o las opciones más útiles para el entorno de pruebas.
- Evaluar las aplicaciones SDN existentes para los problemas de red más relevantes (Ingeniería de tráfico (TE), seguridad, monitoreo y administración).
- Contribuir con al menos tres variantes y/o funcionalidades sobre las aplicaciones SDN que aporten soluciones a los desafíos de las redes modernas, en base a las evaluaciones realizadas.
- Desarrollar una guía de prácticas para el entorno de pruebas que sirva de base para el estudio, la investigación y desarrollo de las SDN en la Universidad de Cuenca.

1.5. Estructura del documento

En este capítulo se ha presentado las ventajas asociadas a las SDN ante la evolución que necesitan las redes tradicionales, anotando los retos dentro de este nuevo enfoque. Lo consiguiente del documento se estructura de la manera que continúa:

- **Capítulo 2:** Evidencia de manera ordenada y descriptiva los principales trabajos relacionados con el presente proyecto; analiza las contribuciones de diferentes autores para abordar los retos presentados, destacando el desarrollo y conclusiones de cada propuesta. Además, se establece las variantes y diferencias que propone este trabajo de titulación con respecto a los planteamientos y soluciones de otros autores.
- **Capítulo 3:** Detalla los fundamentos teóricos del enfoque SDN, de sus componentes, arquitectura y protocolos. Muestra una descripción conceptual y técnica de los elementos que intervienen en este trabajo de titulación, tanto hardware como software.
- **Capítulo 4:** Describe de manera técnica la metodología empleada para el diseño e implementación del entorno de análisis, experimentación, evaluación y desarrollo para redes definidas por software (SDN).
- **Capítulo 5:** Evidencia las pruebas de funcionamiento realizadas y los resultados obtenidos, en el entorno de emulación y entorno de implementación física.
- **Capítulo 6:** Presenta las conclusiones, recomendaciones y trabajos futuros.
- **Anexo A:** Se presentan las guías de prácticas desarrolladas.

CAPÍTULO 2

Estado del arte y trabajos relacionados

En este capítulo se presenta una recopilación y resumen de los principales trabajos de investigación que abordan el estudio de las redes definidas por software (SDN). Se ha considerado en primer lugar trabajos desarrollados en el ámbito nacional y luego a nivel mundial. Los términos clave en la búsqueda tienen el objetivo de evidenciar las implementaciones existentes de *testbeds*, bancos o entornos de pruebas SDN, que involucren el uso de hardware de bajo costo o que, más específicamente, empleen Raspberry Pi e incluyan preferentemente los controladores POX, Ryu y ODL.

2.1. Estudios sobre SDN en Ecuador

Tabla 2.1: Registro de trabajos relacionados con SDN en las principales universidades de Ecuador

Institución	Trabajos que incluyen "SDN"	Son Prototipos de Pruebas SDN	Son Prototipos de Pruebas SDN con RPi
EPN	11	10 de 11	1 de 10
ESPE	8	4 de 7	0 de 4
UG	8	2 de 8	1 de 2
UPS	7	3 de 7	1 de 3
ESPOCH	4	4 de 4	0 de 4
ESPOL	2	0 de 2	–
Universidad Nacional de Loja (UNL)	1	1 de 1	0 de 1
Universidad Central del Ecuador (UCE)	1	1 de 1	0 de 1
Universidad de Cuenca	0	–	–

Las principales universidades del Ecuador cada día trabajan en proyectos y estudios que permitan estar a la vanguardia tecnológica. En el contexto de las SDN, siendo aún una tecnología en despliegue y posiblemente un puente para las NFV, se las puede abordar desde el estudio experimental mediante bancos de prueba, *testbeds*, entornos de emulación, prototipos o cualquier otro enfoque que lleve por

objetivo evaluar y enseñar.

En la Tabla 2.1 se muestra la cantidad de trabajos que registran las principales universidades de Ecuador dentro de sus respectivos repositorios bibliográficos digitales DSpace que incluyen el término clave "SDN" y que refieren a "*Software-Defined Networking*", hasta agosto de 2021 se registran las cantidades que se detallan en la Tabla 2.1.

Dentro de la justificación del presente trabajo de titulación, abordado en 1.2, se han anotado estudios realizados por varias universidades del país y algunos de éstos se contabilizan en la Tabla 2.1. En la Tabla 2.2 se hace un recuento y resumen de los trabajos ya mencionados, donde se puede relacionar la casa de estudios donde se desarrolló, la infraestructura de conmutación en la que se emuló, simuló o implementó, el controlador empleado y una referencia general del campo de estudio en el que se ha desarrollado cada propuesta.

Adicionalmente en la Tabla 2.2 se incluyen algunos trabajos de otras universidades del país que no han sido presentados en 1.2, estos trabajos se describen también a continuación.

Tabla 2.2: Trabajos sobre entornos SDN en universidades de Ecuador, incluye los mencionados en 1.2

Institución	Infraestructura	Controlador	Campo de Estudio
ESPOL	RPi 3B con OVS	Floodlight	Ancho de banda y tiempos de respuesta, SDN vs. red tradicional. [7]
EPN	Emulado en Mininet e implementado en RPi con OVS	Ryu	Tasa efectiva y retardo, simulador vs. real. Adicional *Firewall*. [8]
ESPE	Emulado en Mininet e implementado en Mikrotik	ODL	*Testbed* de red inalámbrica. [9]
ESPOCH	Emulado en Mininet	Ryu	QoS, control de tráfico ICMP. [10]
	Zodiac FX	Ryu	IDS/IPS Snort para protección DoS e interoperabilidad con DHCP, DNS y HTTP [11]
	Tp-Link TL-WR1043ND	Ryu, POX y Pyretic	Comparación de controladores y evaluación de rendimiento en red inalámbrica [16]
UG	RPi con OVS	ODL	Prototipo SD-WAN empleando VXLAN [12]
UPS	Virtualizada con OVS	ODL	NFV para balanceo de carga y seguridad distribuida, con IPv6 [13]
	Mikrotik	ODL	MPLS con políticas de QoS para telefonía IP [14]
	Emulado en Mininet e implementado en Zodiac FX apoyado en RPi	Floodlight, HP VAN SDN, POX, ODL y Ryu	Módulo didáctico para pruebas de prácticas SDN [17]
UNL	Emulado en Mininet e implementado en Zodiac FX	Ryu	Prototipo de entorno de pruebas para prácticas [18].

Por parte de la UNL, se ha implementado un prototipo de SDN emulado en Mininet y que luego se lleva físicamente al conmutador Zodiac FX. Se ha desarrollado un escenario de 10 prácticas, las 5

primeras corresponden a familiarización con Mininet y las 5 restantes involucran el uso del conmutador físico [18]. A nivel de emulador se abordan temas como la instalación de software necesario, *Firewall*, *Router*, IPv6 y transmisión de vídeo; mientras que en implementación física se plantean prácticas sobre introducción al uso del conmutador, funcionalidad de *switch* estándar, monitor de tráfico, *Firewall* y *Router*.

En la ESPOCH se realizó una comparación entre diferentes controladores: Ryu, POX y Pyretic; se consideró parámetros como latencia, rendimiento de la red y facilidad de programación, para este fin se estableció un entorno de prueba y se evaluó con iPerf. Al finalizar, las métricas presentaron a POX como el mejor controlador según los aspectos considerados. Luego, con este controlador, se crea un prototipo SDN como solución a las limitaciones presupuestarias, problemas de congestionamiento y escalabilidad presentes en uno de los *hostpots* de la ESPOCH gestionado por un controlador *Cisco Wireless Local Area Network (WLAN) Controller*. El prototipo podría reemplazar las funciones de la red física y se corroboraría si SDN es una solución para la red inalámbrica. Al implementar el prototipo SDN usando POX, la recopilación de información y comparación de rendimiento de la red existente frente a la propuesta SDN mostró que esta última no era apta, puesto que se logra mejor velocidad de trasmisión y ancho de banda pero se presentaba una pérdida de información considerable [16]. Esta propuesta fallida se componía de 7 controladores POX, antenas, *routers*, servidor Remote Access Dial In User Service (RADIUS) y varios *access point*.

En [19] se diseña una red de sensores inalámbricos (Wireless Sensor Network (WSN)) con SDN para mediciones de radiación solar. Los nodos sensores funcionan con Arduino y transmiten por módulos Xbee hacia nodos centrales implementados en una RPi que funciona de puerta de enlace para enviar las lecturas a una base de datos en Firebase. Se aborda así el concepto de redes de sensores definidas por software (Software-Defined Wireless Sensor Network (SD-WSN)). Las pruebas evidencian la recolección efectiva de las mediciones.

Trabajos más específicos sobre prototipos o entornos de prueba dirigidos al apoyo del aprendizaje o familiarización hacia SDN se han desarrollado en el país. En la UPS, [17] implementa un módulo didáctico conformado por un conmutador Zodiac FX y 3 Raspberry Pi Model 3B para alojar el controlador y 2 hosts; estos elementos fueron organizados en un soporte para fines estéticos, luciendo de manera modular para la implementación de algunas prácticas. Se han generado 10 prácticas, con su respectiva documentación y códigos, respaldadas en un espacio en nube accesible a los estudiantes. Estas prácticas abordan temas introductorios al software y hardware necesario, balanceo de carga, uso de varios controladores (POX, Floodlight, HP VAN SDN, ODL y Ryu), *Firewall*, *Simple Switch*, IDS/IPS y por último, una aproximación a Faucet.

2.2. Principales estudios sobre SDN a nivel mundial que involucran hardware Raspberry Pi

Alrededor del mundo se han desarrollado estudios sobre SDN que involucran a RPi como una solución o herramienta a considerar en el despliegue de redes o bien en la experimentación de las mismas.

En la Universidad Nacional Autónoma de Nicaragua (UNAN), sede León, se implementó un prototipo de SDN y la evaluación de diferentes controladores, con el objetivo de crear un material de apoyo que permita a estudiantes obtener el conocimiento para implementar, analizar e interpretar el

uso y funcionamiento de OpenFlow para SDN. Para cumplir con el objetivo indicado se generó una guía básica que permita entender el uso del protocolo OpenFlow, al igual que se desarrollaron 12 guías de prácticas de laboratorio. Dichas guías fueron elaboradas en un orden secuencial correspondiente al grado de dificultad que presente cada una de ellas; abordando temas puntuales como: diseño de topología, modificación de tablas de flujo, controlador POX, reglas de *Firewall*, controlador SDN Kinetic, lenguaje SDN Pyretic, controlador Floodligth, balanceo de carga y ataque DoS [20].

En la Universidad Politécnica de Valencia (UPV) se realizó un estudio del controlador SDN Ryu sobre una Raspberry Pi Model 4. En dicho trabajo se ejemplifican aplicaciones Ryu, es decir, aplicaciones que vienen instaladas junto con el controlador. Dichas aplicaciones corresponden a archivos Python ejecutables, las aplicaciones implementadas fueron: *Firewall*, *Spanning Tree*, acciones para QoS y monitor de tráfico. Para cada aplicación se crea un entorno SDN en donde, en Raspberry Pi se implementa tanto Mininet como Ryu [21].

En la Universidad de Cantabria (UniCan), España, se realizó un trabajo en el cual se integran técnicas de ingeniería de tráfico (TE) en redes SDN, se proponen dos casos; el primero es un balanceador de carga multicamino y el segundo es la monitorización de tráfico con cierta QoS. La implementación se desarrolla en una máquina virtual (Virtual Box) en la cual se instala Mininet y el controlador Ryu. En el balanceador multicamino se dirige el tráfico generado mediante iPerf hacia distintas rutas de tal manera que no se sature un solo camino. Para el escenario de monitorización de tráfico se suponen dos tipos de tarifas móviles en una compañía, y se hace uso de distintas colas en OpenFlow en las que cada cola corresponde a una fase de la tarifa. Al finalizar el trabajo se demuestra que las SDN permiten una rápida implementación, logrando resolver problemas de una manera más limpia, flexible y eficaz [22].

En [23] se realizó una comparación de 2 soluciones de bajo costo como son Zodiac FX y RPi. El escenario de prueba experimental considera un controlador Ryu en un ordenador para 2 *hosts* conectados al conmutador a ser evaluado. Se ejecuta el controlador con la aplicación *Simple Switch* de Ryu y se evalúa con iPerf, también se realizan pruebas con el conmutador Cisco Catalyst 2950 no-SDN para considerarlo de referencia. Los resultados evidencian que las dos soluciones son capaces de soportar una red doméstica de tamaño promedio otorgando flexibilidad y siendo un entorno de bajo costo amigable para el usuario. Cada uno de los dispositivos presentó características en las que sobresalen y se muestran en la comparativa de la Figura 2.3. Según los aspectos considerados, RPi presenta mayores características a favor, sin embargo, se evidencia su debilidad en la falta de una interfaz gráfica de usuario, Graphical User Interface (GUI), para configurar el conmutador y la complejidad requerida, además que, no presenta un entorno directo para la gestión de Virtual Local Area Network (VLAN).

Tabla 2.3: Características entre los conmutadores [23].

Característica	**Raspberry Pi**	**Zodiac FX**
Precio	✓	
Plug and Play		✓
OpenFlow	✓	
Gestión de VLAN		✓
Gigabit Ethernet	✓	
Puertos	✓	✓
Escalabilidad	✓	
Interfaz Gráfica		✓
Autónomo	✓	
Entradas de tabla de flujo	✓	

Por último, en [1] se implementa una SDN con RPi, donde se ha considerado el uso del controlador Floodlight instalado en un ordenador y dos conmutadores con OVS 2.5 sobre RPi 3. A cada conmutador se conecta un *host*, como se muestra en la Figura 2.1. Se realizan pruebas básicas de conexión ejecutando la aplicación MACTracker que incluye el controlador.

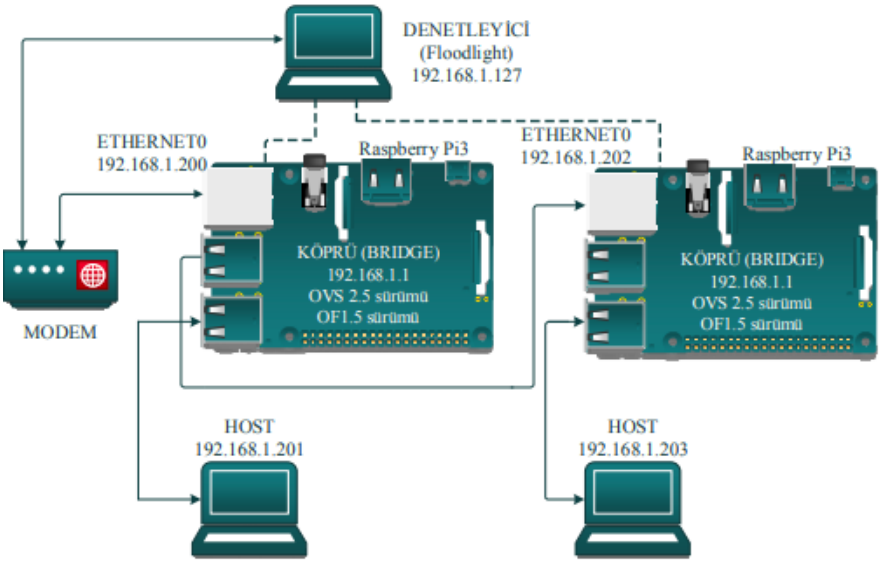


Figura 2.1: Diseño de implementación de una SDN con RPi. Extraído de [1].

De este trabajo vale destacar el uso vanguardista de OpenFlow 1.5, sin embargo, la versión de OVS 2.5 no es la más actual pero si la más estable, compatible y con menores problemas al instalar sobre RPi.

CAPÍTULO 3

Marco teórico

3.1. SDN

Las redes definidas por software (SDN) permiten programar, controlar y administrar directamente los recursos de red facilitando el diseño, entrega y operación de servicios de red, de forma dinámica y escalable.

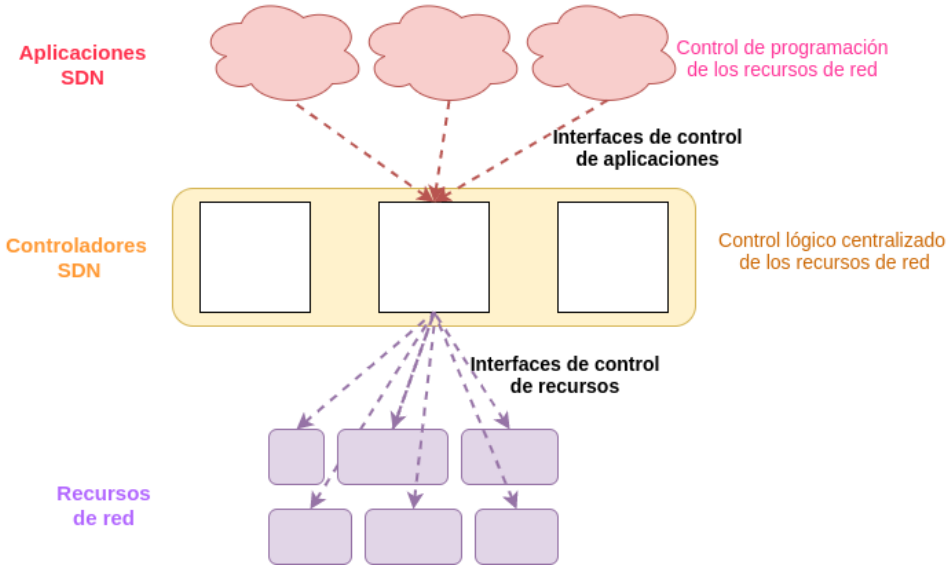


Figura 3.1: Concepto de SDN

La idea fundamental es el desacoplamiento de las dos funciones principales de la capa de red, es decir, del plano de datos (reenvío o *forwarding*) y del plano de control (enrutamiento o *routing*). Así, una SDN tiene todo el control de la red en el controlador SDN que permite controlar, programar y administrar los recursos de red por medio de aplicaciones SDN. El controlador realiza un control lógico centralizado de los recursos que son los encargados de cumplir funciones como transporte y

procesamiento de datos. El controlador proporciona una abstracción de los recursos de red a las aplicaciones SDN a través de la interfaz de control de aplicaciones y modelos de información y datos; las aplicaciones personalizan y automatizan operaciones de los recursos de red a través de dicha interfaz, se indica en la Figura 3.1 [24]. Las ventajas fundamentales de SDN respecto a la red tradicional son las siguientes:

- SDN reduce el tiempo el tiempo de respuesta a diversas solicitudes de usuarios.
- Permite mayor flexibilidad de operaciones de red.
- Adapta los requisitos de conectividad del cliente por medio de una negociación dinámica de las características de servicios de red y control dinámico de recursos.
- Mejora la disponibilidad y eficiencia de recursos de red.
- Permite planificar los recursos en función al número de solicitudes de los clientes, QoS, ingeniería de tráfico (TE) o seguridad.
- Gasto reducido, tanto los gastos de capital (Capital Expenditures (CAPEX)) como los gastos operativos (Operational Expenses (OPEX)).

3.1.1. Capacidades de alto nivel

SDN es capaz de proporcionar las siguientes capacidades:

- Programabilidad: SDN automatiza las operaciones de los recursos de red según las necesidades. La programabilidad permite automatizar la prestación de servicios permitiendo crear un entorno dinámico, al igual que permite combinar diversas funcionalidades que ayuda a la abstracción de recursos.
- Abstracción de recursos: el comportamiento de recursos pueden ser abstraídos o controlados por los programadores debido a los modelos de datos que proporcionan una vista detallada y abstracta de los recursos de red sean estos físicos o virtuales. Los modelos de datos proporcionan una vista abstracta de los recursos de red a las aplicaciones SDN, lo que facilita que los programadores simplifiquen la lógica de sus programas sin tener un conocimiento detallado de recursos y tecnologías de red subyacentes [24].

3.1.2. Arquitectura

SDN posee un arquitectura de alto nivel que consta de tres capas como se muestra en la Figura 3.2.

- Capa de aplicación: Las aplicaciones SDN definen el comportamiento de los recursos de red, estas interactúan con la capa de control SDN por medio de interfaces de control de aplicaciones de tal manera que la capa de control personalice el comportamiento y propiedades de los recursos de red. Para programar una aplicación SDN se usa la vista abstracta de los recursos de red la cual es proporcionada por la capa de control SDN mediante modelos de información y datos. Las aplicaciones son programas que indican comportamientos y recursos al controlador SDN a través de la interfaz API. Una interfaz en dirección norte es la conexión entre el controlador y las aplicaciones.
- Capa de control: Proporciona un medio de control dinámico y determinista del comportamiento de recursos de red según las instrucciones de la capa de aplicación. Las aplicaciones SDN indican cómo controlar y asignar los recursos de red interactuando con la capa de control SDN. La

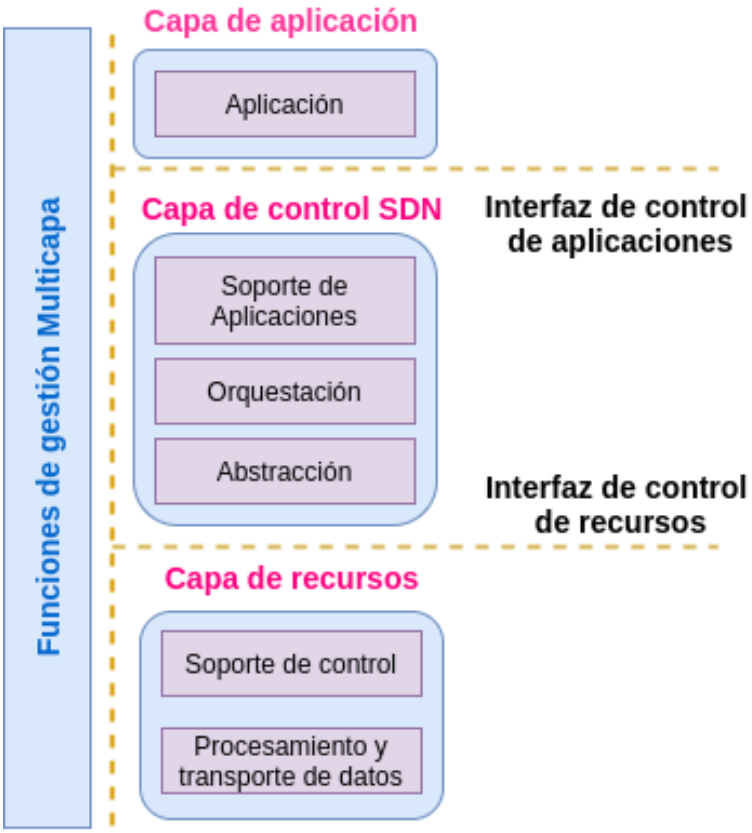


Figura 3.2: Arquitectura SDN.

configuración que se indica en las aplicaciones SDN es abstraída mediante modelos de información y datos, la cual varía según la aplicación.

- Soporte de aplicaciones: Proporciona una interfaz de control de aplicaciones para que las aplicaciones accedan a información de la red y así pueda programar los comportamientos de red.
- Instrumentación: Proporciona el control y gestión de los recursos de red y coordina las solicitudes de la capa de aplicación para los recursos de red. Interactúa con las funciones de gestión de otras capas para proporcionar gestión de usuarios, creación de servicios, entre otras.
- Abstracción: Interactúa con los recursos de red y proporciona una abstracción de los recursos de red, incluyendo capacidades y características de red para admitir la gestión de recursos físicos y virtuales.

- Capa de recursos o plano de datos: En esta capa los elementos de red realizan el transporte y procesamiento de paquetes de acuerdo a las decisiones de la capa de control SDN, dichas decisiones se envían a la capa de recursos por medio de la interfaz de control de recursos. Los dispositivos de red del plano de datos realizan funciones como:
  - Función de soporte de control: Interactúa con la capa de control SDN, el conmutador se comunica con el controlador y éste administra el conmutador por medio del protocolo OpenFlow.

  - Función de reenvío de datos: Proporciona funcionalidades de reenvío y enrutamiento de datos. En el reenvío de datos se manejan los flujos de datos entrantes para reenviarlos a las rutas calculadas de acuerdo con los requisitos definidos en las aplicaciones SDN. El control de reenvío de datos es proporcionado por la capa de control SDN.

  El enrutamiento de datos es proporcionado por el control de red y servicios en la capa de recursos según las reglas que personalice la capa de control SDN para las aplicaciones SDN.
- Funciones de gestión multicapa: Son funciones de capa cruzada que permiten gestionar las funcionalidades de las capas de aplicación, capa de control SDN y capa de recursos. También poseen funcionalidades para gestión de fallas, configuración, contabilidad, rendimiento y seguridad.
- Interfaces: En la arquitectura SDN existen dos tipos de interfaces que proporcionan acceso a los controladores SDN y a los recursos de la red permitiendo un control programable.
  - Interfaz de control de recursos: Esta interfaz es independiente de la tecnología y es usada para la interacción entre la capa de control SDN y la capa de recursos, las cuales intercambian información de cambios en la topología de red, congestión, etc.
  - Interfaz de control de aplicaciones: Se usa para la interacción entre la capa de aplicación y la capa de control SDN, la capa de la aplicación usa esta interfaz para tomar decisiones como asignación de recursos, administración de aplicaciones prioritarias, etc.

Los beneficios de realizar la separación entre el plano de control y el plano de datos permite la facilidad para seleccionar rutas basadas en un conjunto de políticas, pues el controlador es capaz de controlar el estado en el plano de datos de forma independiente. Es decir, sin depender del software, topología de red, o cualquier otra tecnología que implementen los dispositivos del plano de datos. Esta separación también beneficia los centros de datos, pues permite reducir de manera drástica el gasto de ejecutar un centro de datos, el controlador permite adaptar la red para servicios específicos y permite mejorar e innovar de forma rápida. En definitiva, el controlador permite facilitar y flexibilizar la administración y operación de un centro de datos. Actualmente cuando los enlaces se saturan o están inactivos, el redireccionamiento estático es la solución pero este afecta el tráfico y no es completamente eficiente. Las SDN permiten automatizar la red decidiendo el redireccionamiento de tráfico en función a una política dada [25].

3.1.3. Protocolo OpenFlow

En la actualidad los dispositivos de red tienen su propio plano de control y plano de datos para el reenvío de paquetes, por lo que cada dispositivo tiene una perspectiva de la red y que para otro dispositivo pueda verla, debe conectarse a este. El plano de control es el cerebro para cualquier operación, crea tablas base de información de enrutamiento (Routing Information Base (RIB)) o tablas de direcciones de control de acceso al medio (Media Access Control (MAC)), dichas tablas son enviadas al plano de reenvío de datos para ejecutar el envío de tráfico.

OpenFlow es uno de los protocolos utilizados para la comunicación hacia el sur. Es decir, entre el controlador y el *switch* o conmutador, permite que el plano de control interactué con el plano de datos. Mediante un intercambio de mensajes el controlador indica cómo se debe comportar el *switch* al igual que éste informa su configuración y condiciones de tráfico al controlador. Esto permite una mayor funcionalidad y capacidad de programación.

OpenFlow no actualiza las configuraciones de un dispositivo de red, sino que es el encargado de actualizar las tablas de flujo y encontrar el identificador de ruta de datos (Datapath Identifier (DPID)) del conmutador [2].

Un conmutador lógico OpenFlow puede tener una o más tablas de flujo, una tabla de grupo, una tabla de métricas y uno o más canales OpenFlow conectados a un controlador, tal como lo esquematiza la Figura 3.3.

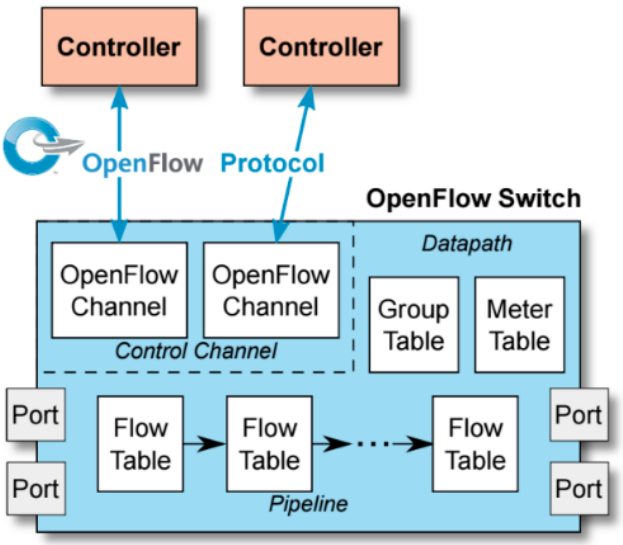


Figura 3.3: Componentes de un *switch* OpenFlow. Extraído de [2]

3.1.3.1. Tablas de flujo

El controlador agrega, actualiza o elimina entradas de flujo en las tablas de flujo, existen entradas de flujos reactivas y pro activas. Las entradas de flujo reactivo son creadas si el controlador aprende de manera dinámica la ubicación de los dispositivos en la topología y actualiza las tablas de flujo en dichos dispositivos para lograr la conectividad de extremo a extremo. Las entradas de flujo pro activo son programadas antes de que llegue el tráfico, es decir, estas son programadas con anticipación.

Una tabla de flujo puede contener varias entradas de flujo; un flujo es un conjunto de paquetes que se transfiere desde un punto final hacia cualquier otro punto de la red. El controlador ve al switch como tablas de flujos, cada flujo se compone de [3]:

- Cabeceras o campos coincidentes (*Match fields*): Identifican el flujo de un paquete, es usado para seleccionar paquetes que coincidan con los valores de los campos.
- Prioridad (*Priority*): Prioridad relativa de las entradas en las tablas de flujo.
- Contadores (*Counters*): Proporciona estadísticas del flujo.
- Campos de acción o instrucciones (*Instructions*): Indica lo que debe hacer el *switch* con los paquetes del flujo.
- Tiempos de espera (*Timeouts*): Tiempo máximo de inactividad antes de que expire un flujo.
- *Cookie*: Usado por el controlador para filtrar estadísticas de flujo, modificación de flujo y eliminación de flujo.
- Indicadores o banderas (*Flags*): Altera la forma en que se administra una entrada de flujo.

En resumen, la estructura de una entrada de tabla de flujo se indica en la figura 3.4 y de una entrada de tabla de grupo se indica en la Figura 3.5.

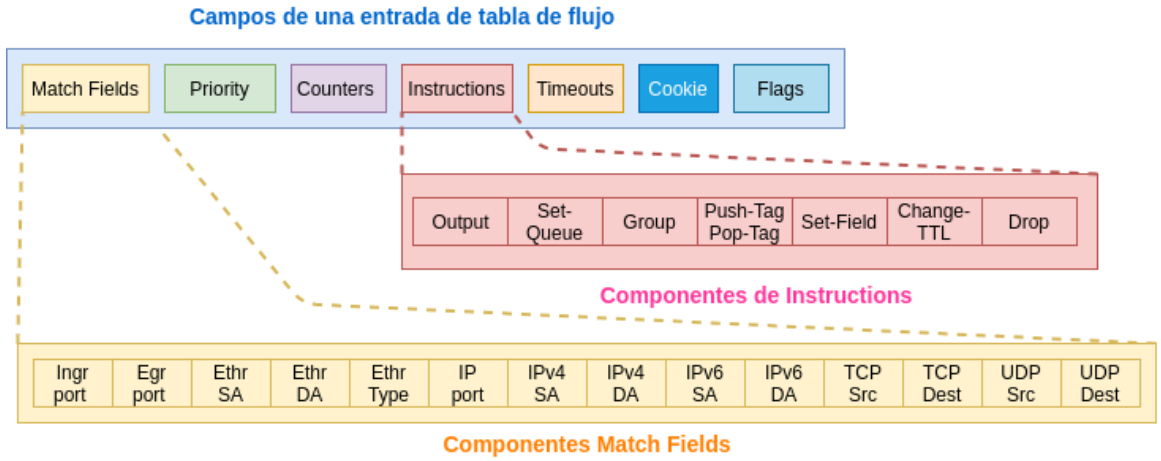


Figura 3.4: Estructura de una entrada de tabla de flujo.

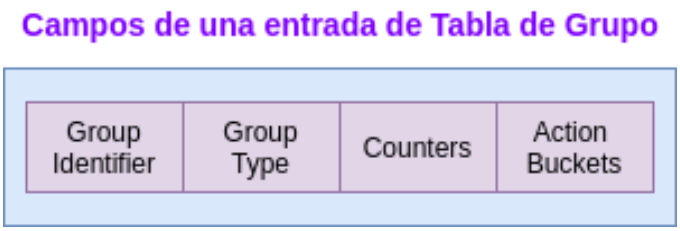


Figura 3.5: Estructura de una entrada de tabla de grupo.

La forma de enseñarle a un *switch* es por medio de reglas OpenFlow. Estas evitan que el switch inunde con preguntas al controlador cada vez que se reciba un paquete; pues mediante las reglas, el controlador indicará al *switch* qué hacer con los paquetes entrantes.

Al llegar el paquete, este comienza por la tabla de flujo 0 y verifica las entradas según la prioridad. Es decir, siempre empieza por la entrada de flujo de mayor prioridad, en caso de que el paquete no encuentre ninguna coincidencia en la tabla 0 y requiera redirigirse a otra tabla lo hace mediante la instrucción `Goto`.

3.1.3.2. Entubado (*Pipeline*)

Un *switch* puede tener una o más tablas de flujo. Cuando existen varias tablas de flujo, estas son organizadas como una tubería (*pipeline*) en la que las tablas están etiquetadas en orden ascendente iniciando desde 0.

OpenFlow establece dos etapas de procesamiento:

- Procesamiento de ingreso: Inicia en la tabla 0 y usa el identificador del puerto de entrada.
- Procesamiento de salida: Ocurre después de determinar el puerto de salida, en esta etapa puede involucrase una o más tablas de flujo. Las tablas con menor número identificador que la de salida, se debe usar como tabla de ingreso.

El procesamiento en tubería inicia en la primera tabla de flujo, el paquete entrante es comparado con las entradas de la tabla de flujo 0. Según sea la coincidencia, pueden existir otras tablas de ingreso. En caso de que exista una coincidencia en una o más entradas, se toma la entrada de mayor prioridad. En caso de que la única coincidencia sea con la entrada de la *table-miss* (entradas de flujo con prioridad 0) esta entrada puede tener acciones de salida, como enviar el paquete hacia el controlador, entonces el controlador permite definir un nuevo flujo. También se puede dirigir el paquete hacia otra tabla de flujo en la tubería; al llegar a la tabla final, ésta debe dirigir el paquete hacia un puerto de salida. [3].

Al usar varias tablas de flujo se permite que un flujo se descomponga en varios subflujos paralelos, como se indica en la Figura 3.6. En ésta se visualiza que, en la Tabla 0 (*Table* 0), una entrada define un flujo para paquetes con una dirección IP de origen y de destino específicas. En la Tabla 1 (*Table* 1) se definen diferentes entradas según el tipo de tráfico (TCP y UDP), y en la Tabla 2 (*Table* 2) se establecen entradas para paquetes del protocolo simple de administración de red (Simple Network Management Protocol (SNMP)), otro dedicado al protocolo para transferencia simple de correo (Simple Mail Transfer Protocol (SMTP)) y uno para el protocolo de transferencia de archivos (FTP).

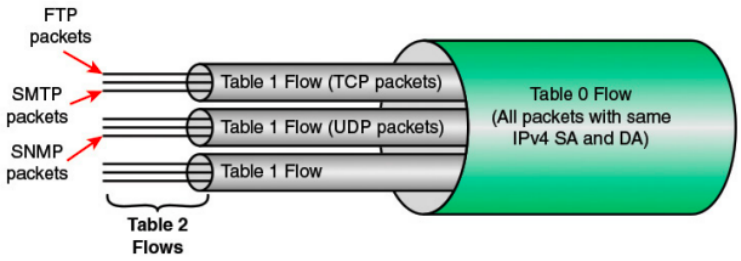


Figura 3.6: Proceso de entubado (*Pipeline*) de OpenFlow. Extraído de [3]

3.1.3.3. Tablas de grupo

Las tablas de grupo permiten que se represente un conjunto de puertos como una sola entidad para reenviar paquetes. Las tablas de grupo consta de 4 componentes, como se observa en la Figura 3.5:

- Identificador de grupo (*Group Identifier*): Es un entero de 32 bits que identifica al grupo, un grupo es una entrada en la tabla de grupos.
- Tipo de grupo (*Group Type*): Determina la semántica y comportamiento del grupo.
- Contadores (*Counters*): Se actualizan cuando los paquetes se procesan en un grupo.
- Depósitos de acciones (*Action Buckets*): Contiene una lista de depósitos (*buckets*), los cuales están compuestos por un conjunto de acciones a ejecutar.

3.1.3.4. Mensajes OpenFlow

OpenFlow maneja 3 tipos de mensajes:

- Controlador a conmutador: Son usados para administrar los conmutadores o *switches* del plano de datos. Estos mensajes pueden ser de los siguientes tipos:
  - *Features*: El controlador solicita el identificador de ruta de datos (DPID) del conmutador.
  - *Configuration*: Establece y consulta parámetros de configuración.
  - *Modify-State*: Agrega, elimina o modifica entradas de flujo.

  - *Read-States*: Obtiene estadísticas.
  - *Packet Out*: Es un mensaje que se envía al *switch* o a un *buffer*.
  - *Role-Request*: Establece el rol del canal OpenFlow.
  - *Asynchronous-Configuration*: Establece un filtro en el mensaje asíncrono que desea recibir en el canal OpenFlow.
- Mensajes asíncronos: Se inician en el conmutador y actualizan al controlador de eventos que se dan en la red y los cambios en el *swicth*.
  - *Packet-in*: Da el control de un paquete al conmutador.
  - *Flow-Removed*: Indica al controlador que se ha eliminado un flujo.
  - *Port Status*: Indica al controlador si se desactiva un puerto de un *switch*.
  - *Error*: Notifica al controlador en caso de que ocurra alguna falla.
- Mensajes simétricos: se inician en el conmutador o el controlador y se envían sin ninguna solicitud.
  - *Hello*: Son mensajes de mantenimiento del canal que se envían entre el controlador y conmutador, surgen en la inicialización del canal OpenFlow.
  - *Echo*: Verifican la vida útil de la conexión, se usan para medir el ancho de banda o latencia.
  - *Experimenter*: Empleados para que los conmutadores ofrezcan funcionalidades adicionales [2].

3.1.4. Controladores SDN

3.1.4.1. POX

Este controlador es uno de los más usados debido a sus componentes y fácil comprensión. A diferencia del controlador ODL, POX permite programas en el lenguaje de programación Python aunque no soporta versiones superiores a OpenFlow 1.0. NOX fue el primer controlados SDN escrito en lenguaje C++, POX es NOX pero en lenguaje Python; la gran ventaja de este controlador es que es muy utilizado, aún mantenido por una comunidad de desarrollo y dispone de soporte actualizado. Entre los componentes que posee POX se distinguen los siguientes:

- Py: Arranca el interprete de Python.
- forwarding.l2_learning: Hace que los *switches* OpenFlow trabajen como *switches* de capa 2.
- forwarding.l2_pairs: Igualmente permite que los *switches* actúen como *switches* de capa 2 e instala reglas basadas en direcciones MAC.
- forwarding.l3_learning: Examina y elabora peticiones así como respuestas para el protocolo de resolución de direcciones (Address Resolution Protocol (ARP)).
- forwarding.l2_multi: Es un *switch* de aprendizaje que usa el componente `openflow.discovery` para aprender toda la topología de la red.
- openflow.spanning_tree: Usa `openflow.discovery` para construir una visión de la topología de la red, elabora un árbol de expansión deshabilitando la inundación (*flooding*) en los puertos que no pertenecen al árbol, de tal manera de evitar la generación de bucles.
- openflow.discovery: Envía mensajes del protocolo de descubrimiento de capa de enlace (Link Layer Discovery Protocol (LLDP)) a los *switches* OpenFlow para descubrir la topología de la red.
- openflow.of_01: Es usado para la comunicación con conmutadores OpenFlow 1.0.
- web.webcore: Inicia un servidor web dentro del proceso POX.

- misc.arp_responder: Se usa para responder las peticiones ARP.
- misc.dns_spy: Supervisa las respuestas DNS y muestra sus resultados.
- misc.mac_blocker: Se usa con aplicaciones de reenvío y abre una interfaz gráfica de usuario que permite el bloqueo de direcciones MAC.
- openflow.debug: Permite que POX cree trazas con mensajes OpenFlow que posteriormente pueden usarse en Wireshark para realizar un análisis.
- openflow.keepalive: Permite que POX envíe solicitudes periódicas *echo* a los *switches* conectados. Adicionalmente, POX posee características como:
  - Interfaz OpenFlow.
  - Compatibilidad con Linux, Macintosh Operating System (Mac OS) y Windows.
  - Soporta las mismas herramientas de NOX.
  - Mejor rendimiento que NOX. [26].

3.1.4.2. Ryu

El controlador Ryu está escrito en Python y es un controlador de código abierto que permite que desarrolladores lleven a cabo nuevas aplicaciones de control y gestión. Ryu admite diversas versiones de OpenFlow desde 1.0 hasta 1.5 [27].

Las aplicaciones le indican al controlador Ryu cómo debe administrar los recursos de red por medio del protocolo OpenFlow. Algunas aplicaciones son instaladas junto con el controlador las misma que permiten a los desarrolladores partir de las mismas para la creación de nuevas aplicaciones. Ryu fue diseñado para incrementar la agilidad de la red y facilitar la administración y adaptación al tráfico [22].

Ryu posee componentes a partir de los cuales se pueden obtener nuevos desarrollos. Los componentes principales son:

- bin/ryu-manager: Se encarga de cargar aplicaciones Ryu, proporciona el contexto a una aplicación y maneja los mensajes que se dan entre aplicaciones.
- ryu.base.app_manager: Maneja las aplicaciones Ryu.
- ryu.controller.controller: Administra las conexiones entre *switches* y se encarga de generar eventos y enviarlos hacia las aplicaciones. Este componente se integra a otros elementos como:
  - `ryu.controller.dpset` que maneja a los conmutadores.
  - El componente `ryu.controller.ofp_event` que define los eventos OpenFlow y,
  - `ryu.controller.ofp_handler` que realiza el manejo básico de OpenFlow.
- ryu.ofproto.ofproto_v1_x_parser: Son módulos desarrollados en Python en los que se hallan el codificador y decodificador para OpenFlow.
- ryu.app.simple_switch: Implementa un *switch* de capa 2.
- ryu.app.rest: Es una API para usar REST.

3.1.4.3. OpenDayLight

El controlador OpenDayLigth (ODL) desarrollado por la Fundación Linux (*Linux Foundation*) tiene la gran ventaja de que puede ser ejecutado en cualquier sistema operativo y hardware, siempre y cuando éste sea compatible con Java.

Este controlador es capaz de adaptarse a diversas topologías y cabe señalar que es el controlador mayormente documentado, el cual constantemente está en desarrollo de actualizaciones y soporte continuo.

ODL usa distintas herramientas que le permitan brindar un mejor rendimiento y buen funcionamiento para redes definidas por software (SDN), entre estas herramientas se destacan:

- Maven: Esta herramienta facilita la automatización de construcción.
- Open Services Gateway initiative (OSGi): permite cargar paquetes y archivos Java Archive (JAR), y vincular paquetes para intercambiar información [28].
- Interfaces Java: Se usan para escuchar eventos, especificaciones y patrones de formación.
- API REST: Se trata de la API por defecto en dirección norte.

OSGi y REST son compatibles con las APIs en la interfaz Norte, API REST está basada en *web* y es usado para aplicaciones que no se ejecutan en el mismo espacio de direcciones que el controlador. Las aplicaciones ejecutan los algoritmos para realizar análisis y orquestar nuevas reglas en la red [28].

La capa de abstracción de servicios (Service Abstraction Layer (SAL)) es aquella que determina cómo brindar un servicio solicitado sin depender del protocolo que se use en el controlador o los dispositivos pertenecientes a la red.

3.2. Open vSwitch

Open vSwitch, denominado también como OVS, es un conmutador virtual multicapa para las principales plataformas de hipervisores, distribuido con licencia Apache 2.0 de código abierto, que busca implementar una plataforma de conmutación de alta calidad. Su diseño permite la automatización masiva de la red a través de un enfoque de programación para sus funciones de reenvío pero que sigue siendo compatible con interfaces de gestión y protocolos estándar como NetFlow, sampled Flow (sFlow), Internet Protocol Flow Information Export (IPFIX), Remote Switched Port Analyzer (RSPAN), Command-Line Interface (CLI), LACP, 802.1ag, entre otros [29].

3.2.1. Funciones

OVS se adapta a entornos de máquina virtual, Virtual Machine (VM), admite múltiples tecnologías de virtualización basadas en Linux. La mayor parte de su código está escrito en Lenguaje C, siendo multiplataforma y fácil de adaptar a otros entornos. La última versión de soporte a largo plazo (Long Term Support (LTS)) de OVS es 2.13.4 mientras que su versión vanguardista es la 2.16.0, la misma que presenta las siguientes funciones [29]:

- Modelo estándar de VLAN 802.1Q con puertos troncales y de acceso.
- Vinculación de tarjetas de red (Network Interface Controller (NIC)) con o sin LACP en el conmutador de flujo ascendente.
- NetFlow y sFlow.
- Configuración de QoS incluido monitoreo.
- Túneles a través de Generic Network Virtualization Encapsulation (GENEVE), Generic Routing Encapsulation (GRE), VXLAN, Stateless Transport Tunneling (STT) y Locator/ID Separation Protocol (LISP).

- Gestión de fallos de conectividad 802.1ag.
- OpenFlow y sus extensiones.
- Base de datos de configuración transaccional con conectores a los lengajes C y Python.
- Reenvío de alto rendimiento mediante un módulo de kernel de Linux, soporte para Linux 3.10 o superior. A partir de Linux 3.3, Open vSwitch se incluye como parte del kernel y el paquete para las utilidades del espacio de usuario está disponible en las distribuciones más populares.

3.2.2. Componentes

El conmutador virtual de OVS incluye los componentes que se listan a continuación y que se evidencian en la Figura 3.7:

- ovs-vswitchd: Demonio que implementa el conmutador, junto con un módulo de kernel de Linux complementario para la conmutación basada en flujo.
- ovsdb-server: Servidor de base de datos liviano que consulta `ovs-vswitchd` para obtener su configuración.
- ovs-dpctl: Herramienta para configurar el módulo del kernel del conmutador.
- Scripts y especificaciones para crear paquetes RPM Package Manager (RPM) para Citrix XenServer y Red Hat Enterprise Linux.
- ovs-vsctl: Utilidad para consultar y actualizar la configuración de `ovs-vswitchd`.
- ovs-appctl: Utilidad que envía comandos para ejecutar demonios Open vSwitch.

3.2.3. Herramientas

OVS siendo un conmutador virtual proporciona las herramientas que se mencionan a continuación:

- ovs-ofctl: Utilidad para consultar y controlar interruptores y controladores OpenFlow, se puede observarla dentro de la arquitectura presentada en la Figura 3.7.
- ovs-pki: Utilidad para crear y administrar la infraestructura de clave pública para conmutadores OpenFlow.
- ovs-testcontroller: Controlador OpenFlow simple que puede ser útil para pruebas, no aplicable en implementaciones reales.
- Parche Tcpdump que permite analizar mensajes OpenFlow.

3.2.4. Funcionamiento

Dos componentes principales son los que dirigen el reenvío de paquetes. Primero se tiene a `ovs-vswitchd`, un demonio en el espacio de usuario que es el mismo, indiferentemente al sistema operativo o entorno. Mientras que el otro componente, un módulo `datapath kernel` o bien, módulo de ruta de datos del núcleo, el mismo que está escrito específicamente para el sistema operativo hospedador asegurando su rendimiento [4].

En la Figura 3.8 se aprecian estos dos componentes, cada uno en su espacio de trabajo, el módulo en el kernel recibe el primer paquete desde una interfaz física NIC o también puede ser desde una tarjeta de red NIC virtual de la VM. En el caso que `ovs-vswitchd` haya instruido al módulo de kernel sobre el tratamiento que debe dar a ese tipo de paquetes, entonces el módulo de ruta de datos sigue las instrucciones, llamadas `actions`, estas acciones dadas por `ovs-vswitchd` detallan los puertos físicos o

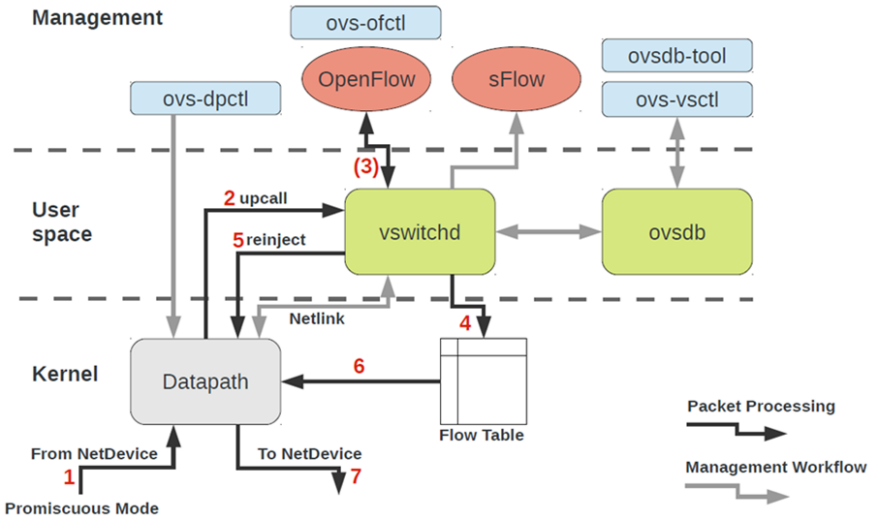


Figura 3.7: Arquitectura de Open vSwitch. Extraído de [4].

túneles en los que transmitir el paquete. Las acciones también pueden especificar modificaciones de paquetes, muestreo de los mismos, o instrucciones para descartarlos.

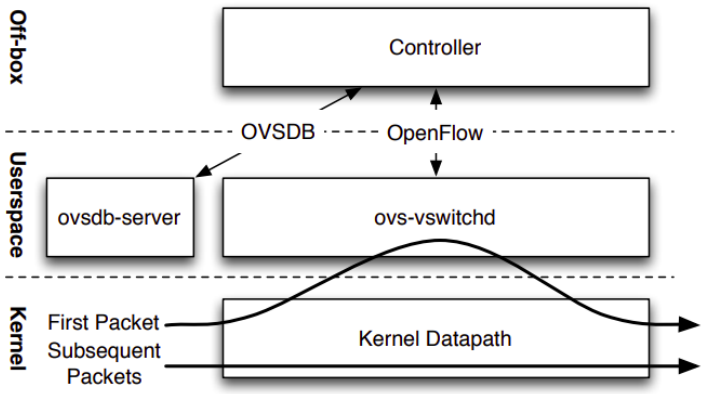


Figura 3.8: Principales Interfaces y Componente de Open vSwitch. Extraído de [4].

Por lo contrario, en el caso en el que la ruta de datos desconoce qué hacer con el paquete, entonces lo envía a `ovs-vswitchd`. En el espacio de usuario este componente determina cómo se debe manejar el paquete, luego regresa el paquete a la ruta de datos indicando el tratamiento que se le debe dar. Comúnmente, `ovs-vswitchd` también le indica a la ruta de datos que se almacene en caché las acciones, para manejar de forma similar a paquetes futuros que presenten parentesco. Este algoritmo se muestra de manera general en la Figura 3.8 y de manera detallada se puede comprender en los pasos etiquetados del 1 al 7 en la Figura 3.7.

OVS también funciona por completo en el espacio de usuario sin la necesidad de un módulo de kernel, que resulta más fácil de implementar que un conmutador basado en un kernel específico. OVS en el espacio de usuario puede acceder a dispositivos Linux o aquellos con kit de desarrollo de plano de datos (Data Plane Development Kit (DPDK)), sin embargo, la implementación sobre dispositivos que no implementan DPDK es experimental y presenta un costo en el rendimiento [29].

3.3. Emulador Mininet

Al trabajar en el diseño e implementación de entornos de prueba de redes definidas por software (SDN) se pueden abordar tres enfoques [30]:

- **Real**: Consiste en trabajar directamente en un banco de pruebas experimental, utilizando dispositivos de red físicos que ejecutan sistemas operativos y aplicaciones instaladas por completo. Los resultados obtenidos pueden ser más realistas, sin embargo, puede ser inaccesible por el alto coste que representa el conjunto de recursos necesarios para la implementación física.
- **Simulador**: A través de la simulación se pretende modelar las interacciones y operaciones de los dispositivos físicos, luego, el modelo planteado se ejecuta por medio de software, una simulación se puede ejecutar a una velocidad mayor o menor que en tiempo real. La simulación presenta ventajas como bajo coste, flexibilidad, escalabilidad, fácil replicación y alto control. Es así que este enfoque se vuelve accesible y recomendable para la mayoría de propuestas de desarrollo e implementaciones. Sin embargo, los resultados pueden diferir en mayor o menor proporción a la realidad, especialmente si el modelo no considera una amplia gama de variables o se realizan asunciones que simplifican el comportamiento; por tanto los resultados deben ser validados o analizados con mayor profundidad para evitar un sesgo de información.
- **Emulador**: Este enfoque se aborda como experimento por lo que se ejecuta en tiempo real y puede involucrar elementos de red que ejecuten sistemas operativos y aplicaciones reales en combinación con otros dispositivos simulados. De este modo se pretende validar los resultados obtenidos desde un modelo hacia una implementación lo más cercana a lo real. Se recae nuevamente en altos costes pero se optimiza el tiempo empleado y se argumenta sólidamente los resultados obtenidos en un simulador.

En SDN es necesario experimentar inicialmente por medio de simuladores o emuladores, a pesar de que el objetivo final lleve hacia una implementación física. Entre simuladores y emuladores más conocidos para SDN se tiene a Network Simulator 3 (NS3), EstiNet y Mininet [30]; en la Tabla 3.1 se comparan estos tres considerando algunas de sus características y prestaciones.

Tabla 3.1: Comparación entre simuladores/emuladores más conocidos para SDN, extracto de [30]

Característica	EstiNet	NS3	Mininet
Modo Simulador	Si	Si	No
Modo Emulador	Si	No*	Si
Compatibilidad con Controladores Externos	Si	No	Si
Soporte GUI	Configuración y Visualización	Solo Visualización	Solo Visualización

***Nota**: Si bien [30] en su recopilación anota que NS3 no posee el modo de emulación para SDN, sin embargo, la página oficial[1] de NS3 muestra documentación referente al modo de emulación a través de un archivo descriptor de dispositivo genérico de red `FdNetDevice` o por medio de otro dispositivo denominado `TapBridge NetDevice`. Con los dos tipos, los nodos actúan como *hosts* "reales" y pueden integrarse a un banco de pruebas, así como en una simulación pueden combinarse elementos simulados

[1] NS3, Emulation Overview: https://www.nsnam.org/docs/models/html/emulation-overview.html

y emulados. Documentación adicional muestra que NS3 tiene soporte para OpenFlow[2]

En este contexto se presenta Mininet como el emulador más utilizado para la investigación de SDN empleando OpenFlow y la programación de procesadores de paquetes independientes del protocolo (Programming Protocol-independent Packet Processors (P4)); principalmente por su compatibilidad con controladores externos reales. Mininet es una herramienta para la creación rápida de prototipos SDN que implementa una red virtual realista con componentes (*switches*, *hosts* y controladores) que ejecutan kernel real y códigos de aplicaciones pero que trabaja en un solo entorno (VM, nube o nativa) para facilitar las pruebas. Esta herramienta está desarrollada, respaldada por una comunidad activa y publicada bajo licencia permisiva *BSD Open Source*.

Mininet permite crear toda una topología a través de tres opciones: línea de comando, interfaz de usuario (User Interface (UI)) interactiva y mediante una aplicación Python. Esto permite interactuar con los nodos de la red, compartir a terceros o implementar en hardware real. Mininet se ha vuelto imprescindible para el campo de desarrollo, enseñanza e investigación [31].

3.3.1. Funcionamiento

Mininet es un sistema orquestador ligero de emulación de red para la creación rápida de prototipos de entorno de red completos. Mininet utiliza la tecnología de virtualización a nivel de sistema operativo de GNU's Not Unix (GNU)/Linux para crear una red virtualizada realista que corre *hosts*, *switches*, *routers* y aplicaciones de red sobre una sola máquina física [32].

Mininet soporta el protocolo OpenFlow lo que permite la emulación de SDN, además utiliza el mecanismo de contenedor provisto por el kernel GNU/Linux para emular nodos en la red. En la Figura 3.9 se muestra el entorno de emulación de Mininet para dos *hosts* conectados a un switch cada uno y estos *switches* se conectan entre sí y hacia un controlador SDN.

Por defecto, todos los *hosts* son procesos regulares que comparten el mismo kernel del sistema operativo, identificador de procesos, nombres de usuario y archivos del sistema. Pero, cada *host* tiene una pila de red independiente con los recursos correspondientes, donde se incluye sus interfaces de red, *caches* ARP y tablas de enrutamiento.

Cada *host* virtual (`vhost 1` y `vhost 2`) puede conectarse a un *switch* virtual (`s1` y `s2` respectivamente; que pueden correr bajo Open vSwitch, descrito en 3.2) gracias a una interfaz virtual y por medio de un enlace virtual con parámetros configurables como: ancho de banda, latencia, tasa de pérdida, etc. La interfaz y enlace virtual son emulados como dispositivo Ethernet virtual GNU/Linux (`veth`). Bajo el mismo fundamento y principio se conectan `vhost 1` y `vhost 2` entre sí.

En la Figura 3.9, el controlador SDN se ejecuta dentro del espacio de nombres `root`, sin embargo, también puede correr fuera de tal espacio, como un controlador remoto.

3.3.2. Beneficios y Limitaciones

El uso de Mininet como plataforma de emulación y experimentación de redes presenta algunos beneficios, los principales se anotan a continuación:

- Dado que es una emulación bastante ligera, acelera el proceso de ejecución, depuración y evaluación. La emulación puede ejecutarse con suficiente escalabilidad en un solo ordenador.

[2] NS3, OpenFlow switch support: `https://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html`

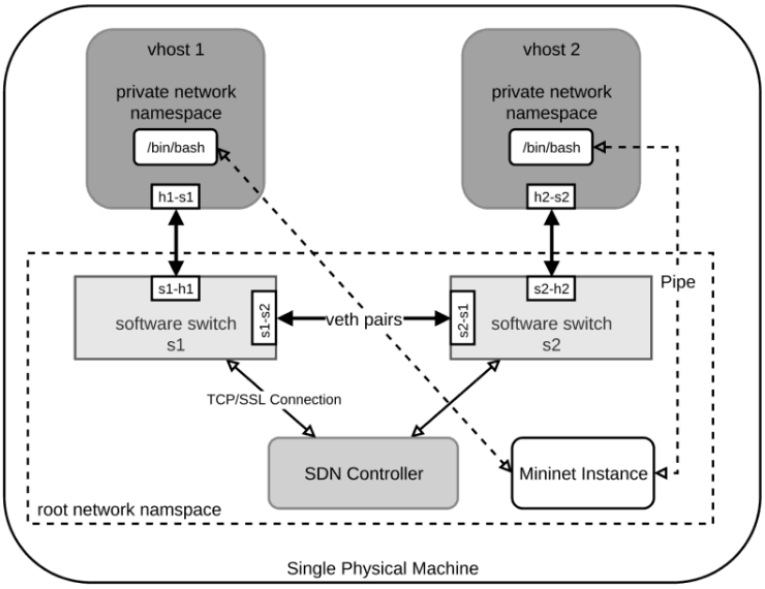


Figura 3.9: Visión de conjunto de Mininet. Extraído de [5]

- Los componentes de la red son completamente personalizables y configurables por medio de la facilidad y entorno amigable que prestan APIs Python.
- Mininet soporta el paradigma SDN por lo que la red emulada así como los *switches* pueden ser programados con las distintas versiones del protocolo OpenFlow.
- La emulación y evaluación a través de *scripts* hace posible compartir el experimento y reproducir fácilmente los resultados.
- Mininet es proyecto de código abierto activo y en desarrollo continuo y cuenta con documentación y tutoriales.

Por otra parte, Mininet presenta algunas limitaciones:

- Todos los componentes virtualizados comparten los recursos de la máquina física, por tanto se debe considerar cuidadosamente las limitaciones que puede presentar el ordenador hospedador.
- El kernel GNU/Linux es compartido y usado por todos los *hosts*, esto no permite que en la emulación se pueda evaluar o comparar el comportamiento de los *hosts* con diferente kernel de sistema operativo o con el mismo kernel pero en diferentes versiones.
- Todos los *hosts* virtualizados residen en diferentes espacios de nombres de red, los demás espacios de nombres, Process ID (PID) y archivos del sistema, son compartidos. Este aislamiento no es suficiente para ejecutar algunas aplicaciones NFV.
- Dado que los archivos de sistema son compartidos por todos los *hosts* virtualizados, los paquetes de *software* requeridos por los *hosts* deben ser instalados en el sistema operativo hospedador. Esto implica cierta dificultad al manejar conflictos de dependencias de *software* y cuando se pretende realizar la configuración de un *host* único que sea fácilmente reproducible.
- Los *hosts* virtualizados por Mininet, por defecto, no soportan el despliegue y administración de aplicaciones contenerizadas como la tecnología Docker. La emulación de aplicaciones que se ejecutan nativamente en nube con movilidad es algo que tampoco está soportado por las implementaciones regulares de Mininet, se necesita de versiones modificadas; así es el caso de

Containernet [33], una bifurcación de Mininet.

3.3.3. Principales Comandos

Mininet es un entorno amplio para la emulación del comportamiento de una red, existen comandos que permiten ejecutar diversas funcionalidades, algunos invocan al entorno de Mininet y otros corren dentro de Mininet, en el Listado 3.1 se anotan y comentan los más relevantes y frecuentes en uso, otros con mayor detalle son mencionados en [34]:

```
user@terminal:~$ sudo mn -c #Clear: Limpia todos los registros temporales.
user@terminal:~$ sudo mn -h #Help: Muestra opciones de inicio de Mininet
mininet> pingall #Realiza ping entre todos los hosts de la topología.
mininet> h1 ping h2 #Realiza ping de manera continua desde h1 hacia h2.
mininet> h1 ping -c 5 h2 #Realiza ping de 5 veces desde h1 hacia h2.
mininet> nodes #Lista los elementos de la red, nodos.
mininet> net #Lista los nodos, enlaces e interfaces conectadas.
mininet> dump #Muestra información sobre los elementos de la red.
mininet> link s1 h1 down #Inhabilita el enlace adyacente entre s1 y h1
mininet> link s1 h1 up #Habilita el enlace adyacente entre s1 y h1
mininet> xterm h1 s1 #Abre una terminal xterm de h1 y otra de s1
mininet> exit #Salir de la CLI Mininet.
```

Listado 3.1: Principales comandos en Mininet

3.3.4. Topologías

La implementación de una topología en Mininet se puede realizar desde una interfaz de línea de comandos (CLI) ejecutando una línea con sintaxis definida por Mininet, cuando se requiere una topología predeterminada. En el caso de una topología personalizada se ejecuta un *script* Python dentro de la CLI; aunque el diseño puede ser realizado utilizado la GUI MiniEdit.

Mininet rápidamente permite emular algunas topologías que trae de manera predeterminada, éstas se han denominado: *Minimal* (Mínima), *Single* (Individual), *Reversed* (Invertida), *Linear* (Lineal), *Tree* (Árbol) y *Torus* (Toroidal). En el Listado 3.2 se anotan las líneas de comando para la implementación de diversas topologías que, en los siguientes apartados se describen, incluyendo el caso de una topología personalizada.

```
user@terminal:~$ sudo mn --topo minimal #Topologia Minimal
user@terminal:~$ sudo mn --topo single,3  #Topologia Single
user@terminal:~$ sudo mn --topo reversed,3  #Topologia Reversed
user@terminal:~$ sudo mn --topo linear,3,1  #Topologia Linear
user@terminal:~$ sudo mn --topo tree,3,2  #Topologia Treee
user@terminal:~$ sudo mn --topo torus,3,3  #Topologia Torus
user@terminal:~$ sudo python miTopologia.py  #Topologia Personalizada
```

Listado 3.2: Lineas de Comando para implementar topologías en Mininet

3.3.4.1. Topología *Minimal*

Esta topología está conformada por un *switch* al que se conectan dos *hosts*. Para implementar esta topología, la sintaxis de la línea de comando que se ejecuta en la CLI se anota en la línea 1 del Listado 3.2. La sintaxis es única y no admite parámetros que configuren la topología. En la Figura 3.10 se pueden observar los enlaces que se crean con los respectivos dispositivos e interfaces involucradas.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Figura 3.10: Salida del comando `net` en Mininet tras ejecutar la topología *Minimal*.

3.3.4.2. Topología *Single*

Esta topología cuenta con un solo *switch* pero en su sintaxis permite la configuración de la cantidad `k` de *hosts* que se conectan, `single,k`. En la línea 2 del Listado 3.2 se anota un ejemplo de implementación de esta topología con $k = 3$, 3 *hosts* conectados a un solo *switch*. Los enlaces creados entre dispositivos con sus respectivas interfaces se muestran en la Figura 3.11

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
```

Figura 3.11: Salida del comando `net` en Mininet tras ejecutar la topología *Single*, k=3.

3.3.4.3. Topología *Reversed*

Se trata de una topología similar a *Single*, cuenta con un solo *switch* y en su sintaxis permite la configuración de la cantidad `k` de *hosts* que se conectan, `reversed,k`. Sin embargo, las conexiones se realizan en orden invertido, en comparación a una topología *Single*. En la línea 3 del Listado 3.2 se anota un ejemplo de implementación de esta topología con $k = 3$, 3 *host* conectados a un solo *switch* con conexiones reversas. Los enlaces creados entre dispositivos con sus respectivas interfaces se muestra en la Figura 3.12.

```
mininet> net
h1 h1-eth0:s1-eth3
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth1
s1 lo:  s1-eth1:h3-eth0 s1-eth2:h2-eth0 s1-eth3:h1-eth0
c0
```

Figura 3.12: Salida del comando `net` en Mininet tras ejecutar la topología *Reversed*, k=3.

3.3.4.4. Topología *Linear*

Esta topología cuenta con `n` *switches* conectados en serie (sin formar ningún bucle) y `k` *hosts* conectados a cada uno de los *switches*, así lo permite configurar en su sintaxis el comando `linear,n,k`. En la línea 4 del Listado 3.2 se anota un ejemplo de implementación de esta topología con $n = 3$ y $k = 1$, 3 *switches* conectados uno a continuación del otro sin cerrar en bucle y a cada *switch* se conecta 1 host. Los enlaces creados entre dispositivos con sus respectivas interfaces se muestra en la Figura 3.13.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3
c0
```

Figura 3.13: Salida del comando `net` en Mininet tras ejecutar la topología *Linear*, n=3 y k=1.

3.3.4.5. Topología *Tree*

En forma de árbol, esta topología implementa `n` niveles de *switches* y `k` ramificaciones en cada nivel, bajo del último nivel se ramifican `k` *hosts*, su sintaxis lo permite bajo la configuración `tree,n,k`. En la línea 5 del Listado 3.2 se anota un ejemplo de implementación de esta topología con $n = 3$ y $k = 2$. Esto implementa 3 niveles de *switches*, cada nivel surge de la ramificación en factor 2 del nivel anterior y bajo el último nivel se conectan 2 *hosts* a cada *switch*. Los enlaces creados entre dispositivos con sus respectivas interfaces se muestra en la Figura 3.14.

```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
c0
```

Figura 3.14: Salida del comando `net` en Mininet tras ejecutar la topología *Tree*, n=3 y k=2

3.3.4.6. Topología *Torus*

Se trata de una topología de mayor complejidad que las anteriores, formando un ordenamiento toroidal de dos dimensiones. Su sintaxis tiene la configuración `torus,m,n`, que refiere a una matriz de

```
mininet> net
h1x1 h1x1-eth0:s1x1-eth1
h1x2 h1x2-eth0:s1x2-eth1
h1x3 h1x3-eth0:s1x3-eth1
h2x1 h2x1-eth0:s2x1-eth1
h2x2 h2x2-eth0:s2x2-eth1
h2x3 h2x3-eth0:s2x3-eth1
h3x1 h3x1-eth0:s3x1-eth1
h3x2 h3x2-eth0:s3x2-eth1
h3x3 h3x3-eth0:s3x3-eth1
s1x1 lo:  s1x1-eth1:h1x1-eth0 s1x1-eth2:s1x2-eth2 s1x1-eth3:s2x1-eth2 s1x1-eth4:s1x3-eth3 s1x1-eth5:s3x1-eth4
s1x2 lo:  s1x2-eth1:h1x2-eth0 s1x2-eth2:s1x1-eth2 s1x2-eth3:s1x3-eth2 s1x2-eth4:s2x2-eth2 s1x2-eth5:s3x2-eth5
s1x3 lo:  s1x3-eth1:h1x3-eth0 s1x3-eth2:s1x2-eth3 s1x3-eth3:s1x1-eth4 s1x3-eth4:s2x3-eth2 s1x3-eth5:s3x3-eth5
s2x1 lo:  s2x1-eth1:h2x1-eth0 s2x1-eth2:s1x1-eth3 s2x1-eth3:s2x2-eth3 s2x1-eth4:s3x1-eth2 s2x1-eth5:s2x3-eth4
s2x2 lo:  s2x2-eth1:h2x2-eth0 s2x2-eth2:s1x2-eth4 s2x2-eth3:s2x1-eth3 s2x2-eth4:s2x3-eth3 s2x2-eth5:s3x2-eth2
s2x3 lo:  s2x3-eth1:h2x3-eth0 s2x3-eth2:s1x3-eth4 s2x3-eth3:s2x2-eth4 s2x3-eth4:s2x1-eth5 s2x3-eth5:s3x3-eth2
s3x1 lo:  s3x1-eth1:h3x1-eth0 s3x1-eth2:s2x1-eth4 s3x1-eth3:s3x2-eth3 s3x1-eth4:s1x1-eth5 s3x1-eth5:s3x3-eth4
s3x2 lo:  s3x2-eth1:h3x2-eth0 s3x2-eth2:s2x2-eth5 s3x2-eth3:s3x1-eth3 s3x2-eth4:s3x3-eth3 s3x2-eth5:s1x2-eth5
s3x3 lo:  s3x3-eth1:h3x3-eth0 s3x3-eth2:s2x3-eth5 s3x3-eth3:s3x2-eth4 s3x3-eth4:s3x1-eth5 s3x3-eth5:s1x3-eth5
c0
```

Figura 3.15: Salida del comando `net` en Mininet tras ejecutar la topología *Torus* $3x3$.

orden $m \times n$ de *switches* conectados entre sí y formando bucles, a cada switch se conecta un *host*. En la línea 6 del Listado 3.2 se anota un ejemplo de implementación de esta topología con $m = n = 3$. Cabe considerar que no se permiten un orden menor a 3×3. En la Figura 3.16 se puede ver esta topología, donde los enlaces creados entre dispositivos con sus respectivas interfaces se muestra en la Figura 3.15.

En esta topología se debe tener precaución al ejecutar aplicaciones que involucran protocolos de descubrimiento de rutas debido a los bucles creados, siendo necesarias implementaciones como el protocolo de árbol de expansión mínimo (Spanning Tree Protocol (STP)).

3.3.4.7. Topología personalizada

La implementación de una topología personalizada requiere de la ejecución de un *script* Python que en su código contiene la configuración de la misma, el desarrollo de este *script* implica la importación de diversos módulos así como una estructuración de clases e instancias de objetos. Este trabajo de diseño puede ser abordado directamente con habilidades de programador o bien se puede usar el GUI de Mininet, MiniEdit. La ejecución de una topología en particular se muestra en la línea 7 del Listado 3.2, en ese caso se trata de un *script* que se ha nombrado `miTopologia.py`.

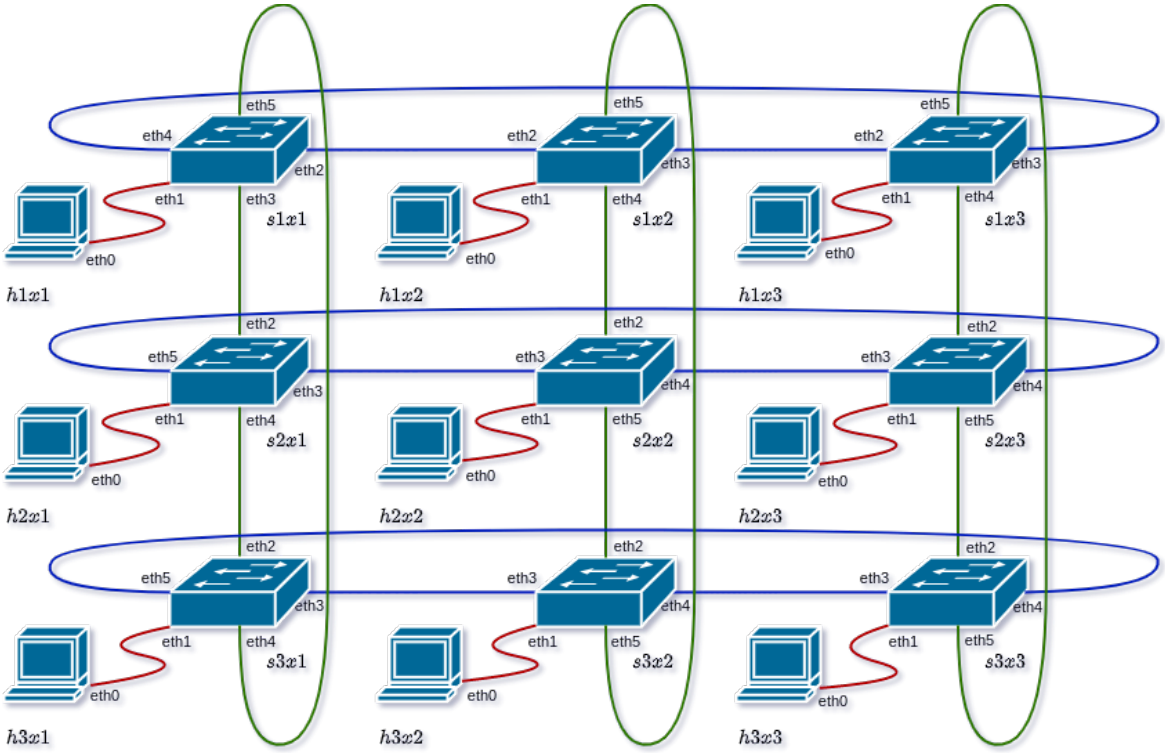


Figura 3.16: Topología Torus $3x3$.

3.4. Raspberry Pi

Raspberry Pi es un producto británico que en contracción se denomina RPi, se trata de un computador, unidad central de procesamiento (Central Processing Unit (CPU)), de bajo costo que nació con fines educativos y didácticos de aprendizaje sobre computación, introducción a la programación y desarrollos electrónicos básicos. Su tamaño lo hace ver como un ordenador de bolsillo por ser muy pequeño pero con un sistema operativo que le faculta capacidades comparables a un ordenador de escritorio [35].

RPi permite la conexión de *hardware* periférico así como la instalación de sistemas operativos de acuerdo a las preferencias y usos que pretenda el programador; además se puede conectar circuitos electrónicos a los pines y controlarlos como entradas y salidas.

Existen diferentes modelos de Raspberry Pi, cada uno levemente diferenciado en sus características, guardando el mismo principio fundamental. En orden de aparición y evolución se registran los siguientes modelos [36]:

- Raspberry Pi 1 A/B/B+
- Raspberry Pi 2 B
- Raspberry Pi 3 B/B+/A+
- Raspberry Pi 4 B

Cada modelo es un computador ligero potenciado por un procesador desarrollado por Advanced RISC Machine (ARM); este procesador es rápido y liviano pero ejecuta *software* bajo la arquitectura Reduced Instruction Set Computer (RISC), que requiere una cantidad menor de transistores que los

procesadores x86 de arquitectura Complex Instruction Set Computer (CISC), considerados como lo más común en ordenadores portátiles. La tecnología de procesadores RISC es preferida para para dispositivos móviles por su bajo consumo energético mientras que los procesadores CISC se consideran convencionales para ordenadores portátiles y de escritorio.

Se tienen diferentes sistemas operativos disponibles donde Raspberry Pi OS luce como el sistema operativo oficial compatible y recomendado para RPi.

La RPi no tiene disco duro, el sistema operativo está instalado en una tarjeta Secure Digital (SD). Se puede interactuar con la RPi mediante el modo consola (por terminal mediante Secure Shell (SSH)) o en el modo gráfico (conectando un monitor a través de High-Definition Multimedia Interface (HDMI)), gracias a los puertos USB se puede conectar periféricos como *mouse* y teclado mientras que para la alimentación de energía cuenta con un puerto micro-USB (o USB tipo C en RPi 4B).

La mayoría de modelos de Raspberry Pi cuentan con conexión Ethernet. A partir de RPi 3 se ha integrado un módulo WiFi y Bluetooth haciendo factible la conexión inalámbrica, en modelos anteriores se necesita de un adaptador USB-WiFi o USB-Ethernet.

3.4.1. Principales modelos y características

En este apartado se describen los dos últimos modelos, que corresponden a aquellos considerados para el entorno práctico de implementación física: Raspberry Pi 3 B+ y Raspberry Pi 4 B.

En la Tabla 3.2 se pone en comparación los dos modelos en mención, aquí se pueden evidenciar notables diferencias en el aspecto de memoria de acceso aleatorio (Random Access Memory (RAM)) así como la capacidad de los códecs de audio y vídeo. RPi 4 B presume mejoras al incluir 2 puertos USB 3.0, la alimentación de energía por USB-C, 2 puertos micro-HDMI con soporte 4Kp60, entre otras características. Ambos modelos se encuentran en producción vigente, sin embargo, luce recomendable el modelo RPi 4 B por presentar memoria RAM de mejores prestaciones a futuro.

3.4.2. Sistema operativo y kernel

Son muchos los sistemas operativos soportados por Raspberry Pi, los más utilizados por sus prestaciones han sido recopilados en [39] y se muestran en la Tabla 3.3 en orden cronológico por su año de publicación. Algunos de estos sistemas operativos están basados en otros más populares, siendo el resultado de una adaptación exclusivamente dirigida para la arquitectura ARM de RPi como es el caso de Raspberry Pi OS, Pidora, Windows IoT Core, SARPi y RetroPie. Otros sistemas operativos, originalmente, estaban diseñados para arquitecturas ARM por lo que pueden ser instalados en una Raspberry Pi y únicamente se han desarrollado paquetes y repositorios para el soporte de funciones propias de RPi, así es el caso de: Kali Linux, Ubuntu Core, RISC OS, Archi Linux ARM y FreeBSD.

De los 10 sistemas operativos presentados en la Tabla 3.3, cabe destacar a Raspberry Pi OS, antes conocido como Raspian, que desde 2015 se presenta como el sistema operativo oficial y recomendando para uso estándar en Raspberry Pi. Este sistema operativo es gratuito, optimizado para RPi y proporcionado a través del gestor de instalación Raspberry Pi Imager. Cuenta con más de 35000 paquetes precompilados para fácil instalación. La comunidad de desarrollo de Raspberry Pi OS se encuentra activa, enfocada en mejorar la estabilidad y rendimiento de los paquetes Debian para RPi [40].

Otros sistemas operativos de terceros presentados en la página oficial de Raspberry Pi [41] son:

Tabla 3.2: Características de Raspberry Pi 3 B+ vs. Raspberry Pi 4 B.

Característica	**Raspberry Pi 3 B+** [37]	**Raspberry Pi 4 B** [38]
Fecha de lanzamiento	14 de marzo de 2018	24 de junio de 2019
Procesador	Broadcom BCM2837B0 Cortex-A53 (ARMv8) 64-bit SoC Cuatro Núcleos	Broadcom BCM2711 Cortex-A72 (ARMv8) 64-bit SoC Cuatro Núcleos
Frecuencia de Reloj	1,4GHz	1,5GHz
Memoria RAM	1GB LPDDR2 SDRAM	2/4/8GB LPDDR4-3200 SDRAM
Conectividad	2.4/5GHz IEEE 802.11b/g/n/ac Bluetooth 4.2 - BLE Gigabit Ethernet sobre USB 2.0	2.4/5 GHz IEEE 802.11ac , Bluetooth 5.0 - BLE Gigabit Ethernet
Cabecera GPIO	40 pines	40 pines
Puerto de Cámara	CSI (para cámara RPi)	Bidireccional MIPI CSI
Puerto de Pantalla	DSI (para pantalla RPi)	Bidireccional MIPI DSI
Puertos HDMI	Uno tipo A	Dos tipo C (soportan 4Kp60)
Audio y vídeo	Puerto estéreo de 4 polos con vídeo compuesto	Puerto estéreo de 4 polos con vídeo compuesto
Multimedia	H.264 (Codificación 1080p30) H.264, MPEG-4 (Decodificación 1080p30) OpenGL ES 1.1, gráficos 2.0	H.265 (Decodificación 4Kp60) H.264 (Decodificación 1080p60, codificación 1080p30) OpenGL ES, gráficos 3.0
Puertos USB	Cuatro USB 2.0	Dos USB 2.0 Dos USB 3.0
SO y almacenamiento	Ranura micro SD	Ranura micro SD
Alimentación de energía	5V/2,5A CC en micro-USB 5V/2,5A CC en pines GPIO	5V CC min. 3A en USB-C 5V CC min. 3A en pines GPIO
Soporte PoE	Habilitado, requiere PoE HAT	Habilitado, requiere PoE HAT
°T de Operación	$0 - 50°C$	$0 - 50°C$
Vigencia de Producción	Al menos hasta enero de 2023	Al menos hasta enero de 2026

- **LibreElec**: Una distribución de Kodi Entertainment Center.
- **Ubuntu Desktop**: Sistema operativo en versión de Escritorio y de código abierto con aplicaciones de utilidad en el hogar, escuela y trabajo.
- **Ubuntu Server**: Distribución popular de Linux para entornos de centros de datos y almacenamiento en la nube.
- **Ubuntu Core**: Distribución de Ubuntu para entornos integrados con optimización en seguridad y actualizaciones confiables.
- **RetroPie**: Sistema operativo enfocado a convertir una RPi en una máquina de juegos retro.
- **TLXOS**: Distribución de prueba gratis de 30 días basado en Debian de ThinLinX.

Desde que Raspbian se oficializó como recomendado para RPi, este sistema operativo ha pasado por una constante y significativa evolución según la versión Debian de la cual estaba basado, así como el kernel de Linux que utiliza. Además que, actualmente, se lo conoce como Raspberry Pi OS, esta evolución se presenta en la Tabla 3.4 de manera cronológica.

A partir de Noviembre de 2018, Raspbian se renombró a Raspberry Pi OS; siendo un nuevo proyecto fue dividido en tres distribuciones separadas, que en esencia comparten el mismo núcleo de sistema operativo. Es así, que se tiene las distribuciones Ligera (*Lite*), Mínima (*Minimal*) y Escritorio Completo

Tabla 3.3: Recopilación de Sistemas Operativos para RPi [39]

Sistema Operativo	Año	Desarrollador	Basado en
RISC OS	1989	ROOL	Arthur
Free BSD	1993	FreeBSD Project	BSD
Arch Linux ARM	2010	Arch Linux Project	Arch Linux
Raspbery Pi OS (antes Raspbian)	2012	Mike Thompson, Peter Green	Debian
SARPi	2012	SARPi Team	Slackware ARM
Kali Linux	2013	Offensive Security	Debian
RetroPie	2013	RetroPie Project	Raspbian
Pidora	2014	CDOT	Fedora
Ubuntu Core	2014	Canonical	Ubuntu
Windowa 10 IoT Core	2015	Microsoft	Windows 10

Tabla 3.4: Evolución, desde Raspbian a Raspberry Pi OS [35]

Año	Basado en	Kernel Linux
2014	Debian 7 (Wheezy)	3.18
2015	Debian 8 (Jessie)	4.1, 4.4 y 4.9
2017	Debian 9 (Stretch)	4.9
2018	Debian 10 (Buster)	5.4 LTS

(*Full Desktop*). La principal razón de esta división se debe al aumento general del tamaño de las distribuciones Raspbian tras su mejora continua, en pocos años su tamaño pasó de 1GB a 1.8GB. Esta división de 3 distribuciones ha permitido las siguientes características:

- ***Lite***: Pesa alrededor de 350MB, no está recomendada para todos los usuarios, sino más bien para usuarios avanzados. Esta versión no incluye software alguno de lo acostumbrado en las versiones Raspbian. No posee un escritorio gráfico lo que la hace una versión bastante ágil y eficiente de Raspberry Pi OS, ideal para ejecutar Servidores Pi sin periféricos.
- ***Minimal***: Pesa alrededor de 1GB, es la versión recomendada por defecto para la mayoría de usuarios nuevos de RPi. Esta incluye el escritorio PIXEL, Navegador Web Chromium, Reproductor VLC y Python. No incluye LibreOffice, Scratch, Sonic Pi y otros programas habituales en Raspbian.
- ***Full Desktop***: Pesa alrededor de 1.8GB, esta versión contiene todo lo que se encontraba en Raspbian: Escritorio PIXEL, LibreOffice, Chromium, VLC, juegos, recursos de programación y muchas más aplicaciones, incluso *Mathematics* de *Wolfram Alpha*.

La selección de la distribución de Raspberry Pi OS depende de los planes de uso que se pretenda para la Raspberry Pi. Si bien *Raspberry Pi Foundation* recomienda la versión *Minimal* para nuevos usuarios; por otra parte, revistas y foros basados en experiencia recomiendan la versión *Full Desktop*

CAPÍTULO 4

Diseño e implementación del banco de pruebas.

En este capítulo se describe de manera técnica la metodología empleada para el diseño e implementación del entorno de análisis, experimentación, evaluación y desarrollo de redes definidas por software SDN. Dicho entorno se compone de prácticas para un escenario de emulación SDN, se presentan los temas e implementaciones que se abordan. Adicionalmente se detallan las características del escenario de implementación física con hardware asequible.

4.1. Descripción general del entorno de emulación

Los estudios de tecnologías emergentes y en especial de redes modernas, en primera instancia, deben evaluarse en entornos controlados para luego ser llevados a implementaciones de menor escala y finalmente a producción, en entornos de mayor escala en escenarios reales. Es por ello que este entorno comienza trabajando a nivel de emulador, procurando ser un laboratorio en lo más didáctico posible, para el aprendizaje de propios y terceros involucrados a este estudio. Es así que este entorno fundamenta su diseño mediante prácticas a estudiar y desarrollar secuencialmente en cuanto a su complejidad y conocimientos por adquirir.

Las prácticas desarrolladas a lo largo de este trabajo engloban una guía de desarrollo de las mismas y la verificación de su correcto funcionamiento, indicando algunos de los resultados esperados. Dichas prácticas tienen el objetivo de evaluar opciones de controladores (Ryu, POX y ODL), evaluar aplicaciones existentes y contribuir con variantes de aplicaciones SDN que sean de utilidad en desafíos de las redes modernas.

Específicamente, se plantea el desarrollo de 12 prácticas que abarcan diferentes temas como la inicialización de controladores SDN, Mininet, agregación de enlace, ingeniería de tráfico (TE), seguridad, monitoreo, administración, VXLAN, BGP, entre otros. A continuación se presenta una descripción general de las prácticas propuestas:

- La **Práctica** 1 que lleva el nombre de “Introducción a escenarios de simulación SDN: Interacción entre el plano de datos y el plano de control”, permite involucrarse con los controladores Ryu,

POX y ODL conectando el plano de datos con el plano de control.

- La **Práctica** 2 titulada "Topologías básicas y personalizadas en Mininet", busca que el estudiante desarrolle destrezas en el emulador de Mininet para ejemplificar topologías predeterminadas así como topologías personalizadas.
- La **Práctica** 3 denominada "Implementación de Agregación de Enlace con Ryu" se fundamenta en una aplicación existente donde, el ejemplo, trabaja con una topología básica (4 *hosts* conectados a un conmutador) y se realiza la configuración de un grupo de enlace (Unión de 2 NIC virtualizadas para funcionar como una sola) para evidenciar la distribución de tráfico por dicho grupo. La práctica plantea configurar la aplicación existente para ejecutarla en una topología de mayor complejidad (3 conmutadores y 4 *hosts*) donde se propone la configuración de 2 grupos de enlace para proceder a un análisis de la distribución de tráfico con mayor rigurosidad.
- La **Práctica** 4 con el nombre de "Ingeniería de Tráfico con Ryu: Manejo de Tablas de Grupo para división porcentual de tráfico" busca dividir el tráfico de manera porcentual hacia dos rutas diferentes, haciendo uso de tablas de grupo y bajo el tipo grupo SELECT.
- La **Práctica** 5 denominada "Ingeniería de Tráfico con Ryu: Manejo de Tablas de Flujo y de Grupo para Tratamiento diferenciado de Flujos TCP y UDP", permite direccionar el tráfico por dos rutas distintas dependiendo el Protocolo de Transporte (TCP o UDP), es decir, el tráfico TCP cursará por una ruta diferente al tráfico UDP.
- La **Práctica** 6 denominada "Ingeniería de Tráfico con Ryu: Redireccionamiento de Tablas de Flujo (*Pipeline*) para clasificación de tráfico", permite desarrollar destreza en el manejo de varias tablas de flujo para clasificar el tráfico según el protocolo, dividiendo un flujo en otros flujos menores y paralelos, de modo que cada uno tenga un tratamiento diferenciado.
- La **Práctica** 7 con el nombre de "Descubrimiento de rutas basado en ARP" pretende evaluar el comportamiento de una aplicación existente y documentada que presente diversas funcionalidades, una de ellas es el descubrimiento de rutas mediante el procesamiento de mensajes ARP. Abordar su estudio permitirá desarrollar la Práctica 8.
- La **Práctica** 8 con el nombre de "Reenrutamiento Rápido" requiere de la comprensión y desarrollo de la práctica antecesora, se busca desarrollar una aplicación de cierta complejidad que permita instalar rutas de respaldo que funcionen ante la caída del enlace de la ruta principal.
- La **Práctica** 9 denominada "Firewall con Ryu", lleva al estudiante a relacionarse con el establecimiento de reglas de seguridad que permitan o bloqueen el tráfico según la condición especificada.
- La **Práctica** 10 denominada "Seguridad con ODL: Firewall y AAA", incursiona en la administración y seguridad sobre el controlador ODL, de tal manera que el estudiante cree un usuario que pueda ingresar al controlador y realizar cambios significantes. De la misma manera se implementará un *Firewall* simple usando NIC.
- La **Práctica** 11 con el nombre de "Compatibilidad de BGP ODL a Routers BGP Tradicional", permite que el estudiante interconecte una red SDN con una red tradicional utilizando el protocolo BGP.
- La **Práctica** 12 denominada "VXLAN", tiene como objetivo crear un túnel VXLAN que permita la interconexión entre dos servidores SDN ubicados en distintos lugares, para la complejidad requerida los dos servidores y su red de dominio se ejecutan en máquinas virtuales, mientras que en la máquina hospedadora se ejecuta el controlador.

Cada práctica ha sido documentada en una guía estructurada con los siguientes puntos:

- Título, opcionalmente subtítulo, que identifica la práctica y el tema específico a tratar.
- Introducción, que especifica a detalle el propósito, fundamento y motivación del tema a abordar. Además se incluye especificaciones de requerimientos tecnológicos, de conocimiento y *software* para el desarrollo de la práctica.
- Marco teórico a desarrollar, enlista temas mínimos pero fundamentales para la comprensión del sustento, justificación y utilidad de la práctica, además, este marco teórico servirá para comprender aspectos técnicos del tema de la práctica.
- Instrucciones, es la parte estructural de la práctica que describe los pasos necesarios, ya sea para modificar una aplicación existente o para desarrollar una nueva por completo. En cada una se presenta, principalmente, la topología a emular y la aplicación a ejecutar, en algunos casos son necesarios *scripts* o configuraciones necesarias que se detallan en esta sección.
- Verificación de funcionamiento y resultados esperados, es un instructivo más, que plantea los puntos a considerar y la forma de ejecutar el entorno de emulación a fin de evaluar correctamente y obtener los resultados deseados.
- Preguntas y resoluciones, este apartado plantea preguntas a responder por el estudiante, con el fin de consolidar el conocimiento y controlar que el trabajo sea realizado a cabalidad.
- Formato del informe, que indica la estructura mínima y recomendada para presentar el trabajo desempeñado por el estudiante y los resultados obtenidos tras desarrollar la práctica, este formato se plantea considerando la calidad académica que sustenta y demanda la Universidad de Cuenca.

4.2. Diagrama de implementación física de costo asequible

Dentro del estudio de redes definidas por software (SDN), el propósito es crear un entorno de análisis, experimentación y desarrollo, no solo a nivel de emulación sino también de implementación física. Esto permitiría dar un paso adelante, dejando el entorno de emulación para abordar pruebas en escenarios de menor escala, aún controlados pero que presentan comportamientos reales como pérdidas de velocidad de ancho de banda y limitaciones de procesamiento.

4.2.1. Hardware considerado para la implementación física

El entorno a considerar debe permitir la implementación física de, al menos, las prácticas más fundamentales y didácticas, sin representar un alto coste. Es por ello que se ha considerado el uso del *hardware* Raspberry Pi Model 3B+ y Model 4B, ordenadores portátiles personales y otros complementos menores como adaptadores USB-Ethernet y cables Ethernet RJ45.

Las 2 unidades Raspberry Pi $3B+$ con las otras 2 unidades Raspberry Pi 4B han sido consideradas para el plano de datos, sus especificaciones técnicas han sido presentadas en el Capítulo 3. OVS permite convertir estos mini ordenadores en *switches* OpenFlow después de algunas configuraciones exhaustivas. RPi cuenta con solo una interfaz de red Ethernet por lo que estaría limitada a funcionar como *switch*; sin embargo, sus puertos USB permiten la escalabilidad mediante adaptadores USB-Ethernet. Es decir, RPi como *switch* puede contar con una cantidad de puertos Ethernet según lo permita la cantidad de puertos USB y la capacidad de procesamiento.

Los ordenadores portátiles disponibles juegan un papel fundamental como terminales de acceso a la

red y uno de ellos para ejecutar el controlador, procurando que el ordenador seleccionado tenga la mayor capacidad de procesamiento posible. La conexión TCP entre el plano de datos y el plano de control se da mediante WiFi en red local.

La Tabla 4.1 presenta un resumen consolidado del *hardware* seleccionado para la implementación física:

Tabla 4.1: *Hardware* seleccionado para el entorno de implementación física

Cant.	***Hardware***	**Función**	**Características**
2	Raspberry Pi Model 3B+	Switch OpenFlow	Ver Tabla 3.2
2	Raspberry Pi Model 4B	Switch OpenFlow	Ver Tabla 3.2
7	Adaptador USB-Ethernet ANERA AE-LU20-2	Puerto Ethernet	Compatibilidad con Linux 2.4 o superior
3	Adaptador USB-Ethernet EVL UE-208B QTS-LANSR9900	Puerto Ethernet	Compatibilidad con Linux 2.4 o superior
1	Ordenador portátil HP	Controlador o terminal	8GB de RAM, Procesador AMD Ryzen 5 3500U con Gráfico Radeon Vega 8, Almacenamiento de 128GB M.2 SSD + 1TB SATA HDD, SO Ubuntu 20.04 LTS
1	Ordenador portátil DELL	Controlador o terminal	32GB de RAM, Procesador Intel Core i7 8th Gen, Almacenamiento de 2TB SATA HDD, SO Ubuntu 18.04 LTS
1	Ordenador portátil LENOVO	Terminal	4GB de RAM, Procesador Inter Core i7 6th Gen, Almacenamiento de 1TB SATA HDD, SO Ubuntu 20.04 LTS
15	Cable Ethernet	Enlace físico	RJ45 de 1/2 metro

En la Figura 4.1 se observa físicamente el contenido que incluye el paquete completo de Raspberry Pi 4B, dos de estos conjuntos han sido adquiridos para la implementación física.

Por defecto, cada conjunto trae los siguientes componentes básicos:

- Placa del ordenador Raspberry Pi 4B.
- Tarjeta microSD de 16GB.
- Fuente de alimentación.
- Carcasa protectora blanca.
- Disipadores de calor.
- Cable HDMI.
- Manual de usuario.

Y de manera complementaria, cada conjunto incluye los siguientes componentes, los mismos que permiten funciones adicionales.

- Ventilador.
- *Mouse*.
- Teclado.
- Conexión de control de encendido.

De manera similar se cuenta con dos conjuntos de Raspberry Pi 3B+, los mismos que incluyen componentes idénticos a los presentados para RPi 4B. Este conjunto se puede ver en la Figura 4.2.

Figura 4.1: Conjunto Raspberry Pi 4B.

Figura 4.2: Conjunto Raspberry Pi 3B+.

Otra parte fundamental para la implementación física son los adaptadores USB-Ethernet, los mismos que han sido seleccionados considerando su coste y compatibilidad. Debido a la limitada disponibilidad de *stock* en la ciudad Cuenca (Ecuador), se han adquirido adaptadores de dos modelos, anotados en la Tabla 4.1.

En la Figura 4.3 se visualiza la presentación comercial de los adaptadores adquiridos.

Figura 4.3: Adaptadores USB-Ethernet adquiridos.

4.2.2. Soporte estructural del entorno de implementación física

Con fines estéticos y de presentación se ha considerado el diseño de un soporte de madera que aloje los elementos de la implementación física: conmutadores, adaptadores USB-Ethernet y que permita la fácil conexión entre los mismos. A continuación se presentan los componentes y la visión en conjunto de la estructura diseñada:

- **Falso fondo**: Para cableado y elementos de alimentación de energía, se observa en la Figura 4.4.

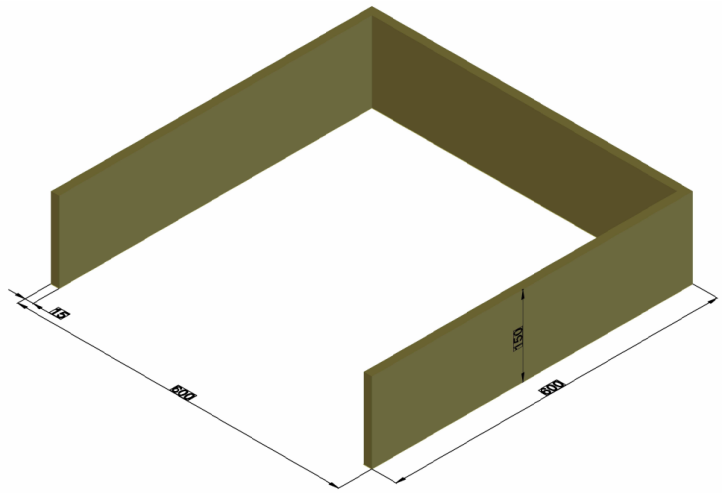


Figura 4.4: Diseño del falso fondo de la estructura de soporte.

- **Base**: Cuenta con orificios colocados en lugares específicos para el paso de cables de alimentación, se observa en la Figura 4.5.
- **Divisiones laterales**: Cuenta con orificios colocados en lugar específicos para el paso de cables de conexión Ethernet entre los conmutadores y ordenadores, se observa en la Figura 4.6

La estructura de soporte, vista en su conjunto se presenta en la Figura 4.7.

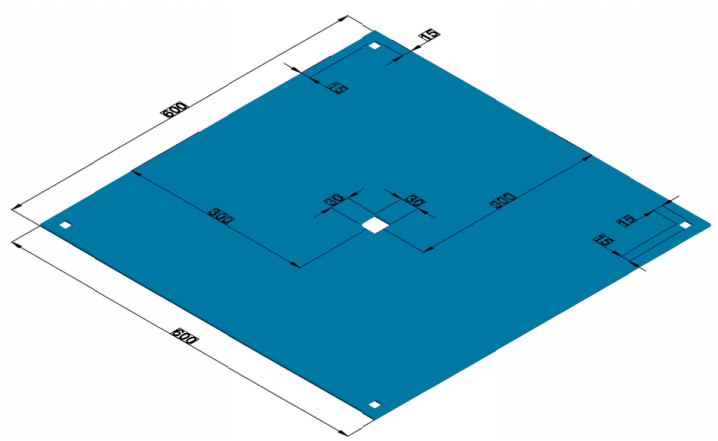


Figura 4.5: Diseño de la base de la estructura de soporte.

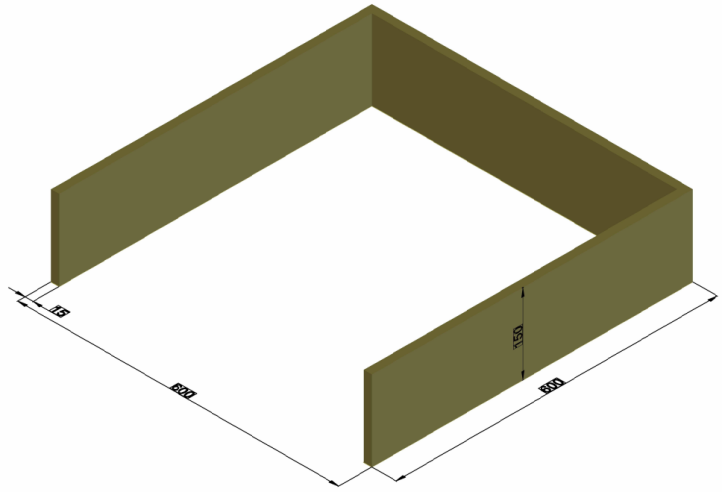


Figura 4.6: Diseño de las divisiones laterales de la estructura de soporte.

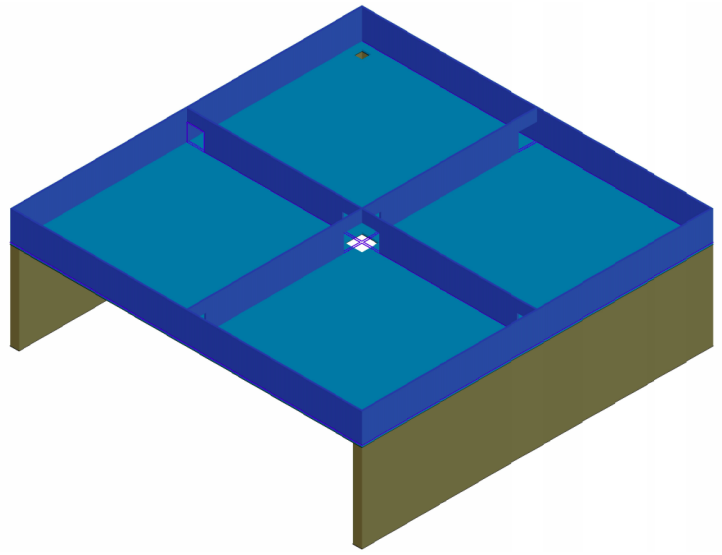


Figura 4.7: Diseño de estructura de soporte, visión en conjunto.

4.2.3. Presupuesto

La inversión realizada para la implementación física contempla el valor monetario de USD 587,96, necesarios para la adquisición de los diferentes componentes. Las adquisiciones de los mismos están dadas a precios de mercado al por menor y se ha procurado la eficiencia económica, sin embargo, estos precios pueden variar dependiendo la temporada del año, disponibilidad de los componentes, lugar de importación y compra, etc.; estos valores se resumen a continuación:

Tabla 4.2: Presupuesto para la implementación Física

Cant.	Componente	P. Unit.	Total
2	Raspberry Pi Model 3B+	79.99	159.98
2	Raspberry Pi Model 4B	149.99	299.98
7	Adaptador USB-Ethernet ANERA AE-LU20-2	9.00	63.00
3	Adaptador USB-Ethernet EVL UE-208B QTS-LANSR9900	7.50	22.50
15	Cable Ethernet RJ45	1.50	22.50
1	Estructura soporte en madera	15.00	15.00
1	Material de papelería	5.00	5.00
		Total	**587.96**

En este presupuesto no se ha considerado los siguientes rubros:

- Transporte: Varía de acuerdo a la localidad y modalidad de transporte que se emplee.
- Alimentación: Por el mismo motivo que el rubro anterior, es variante.
- Ordenadores portátiles personales: Se considera que el grupo investigador o equipo de estudiantes poseen sus ordenadores, por lo que no recurren a nuevas adquisiciones.
- Consumo de energía eléctrica: El presente trabajo no pretende estudiar eficiencia energética ni profundiza en la factibilidad económica.

El valor económico de aproximadamente USD 600.00 es cubierto por los autores de este Trabajo de Titulación, en partes iguales.

4.3. Configuración Previa de Raspberry Pi para implementación del plano de datos

Dentro de la implementación física se ha considerado el uso del *hardware* Raspberry Pi como elemento principal del plano de datos, para tales fines se debe configurar estos ordenadores como *switches* OpenFlow mediante Open VSwitch.

Esta configuración se compone de 3 procesos que se anotan a continuación y se describen a detalle en apartados posteriores:

- Instalación de Open vSwitch sobre Raspberry Pi.
- Configuración de Open vSwitch sobre Raspberry Pi.
- Configuración del *switch* OpenFlow sobre Raspberry Pi empleando Open vSwitch.

Para la configuración, se puede acceder directamente a una terminal de Raspberry Pi mediante su entorno gráfico (usando monitor por HDMI, *mouse* y teclado conectado a RPi), sin embargo, por comodidad se recomienda habilitar la interfaz SSH en la Raspberry Pi, ingresando a:

`Preferencias > Configuraciones de Raspberry Pi > Interfaces`. El usuario por defecto es `pi` y la contraseña `raspberry`. Para ello, la RPi y el ordenador deben estar conectados al mismo dominio de red de área local (Local Area Network (LAN)), que puede ser WiFi.

En el caso de esta implementación física, se abre una terminal y se digita `ssh user@direccionIP` donde el usuario por defecto, `pi`, se ha conservado en todos los *switches* mientras que la dirección IP corresponde a la asignada por la red WiFi. Así también, para identificar a cada *switch* se le ha denominado con letras griegas mayúsculas, como lo muestra la Tabla 4.3. Se ha cambiando el valor de `Hostname` en `Preferencias > Configuraciones de Raspberry Pi > Sistema`.

Tabla 4.3: Resumen de *switches* con sus identificadores y direcciones IP

Símbolo	Nombre	IP	Contraseña
O	Omicron	192.168.1.195	sdno
K	Kappa	192.168.1.200	sdnk
Δ	Delta	192.168.1.191	sdnd
I	Iota	192.168.1.192	sdni

4.3.1. Instalación de Open vSwitch sobre Raspberry Pi

En el Listado 4.1 se anotan los comandos y líneas a ejecutar para la instalación de Open vSwitch sobre RPi, los pasos que se siguen se describen a continuación:

```
1  sudo apt-get update
2  sudo apt-get upgrade
3  sudo su
4  wget https://www.openvswitch.org/releases/openvswitch-2.13.4.tar.gz
5  tar -xvzf openvswitch-2.13.4.tar.gz
6  apt-get install python-simplejson python-qt4 python-twisted-conch automake
       autoconf gcc uml-utilities libtool build-essential pkg-config
7  apt-cache search linux-headers
8  apt-get install linux-headers-rpi
9  cd openvswitch-2.13.4
10 ./boot.sh #Inica el archivo de arranque de la instalación
11 dpkg-checkbuilddeps
12 apt-get install graphviz debhelper dh-autoreconf python3-all python3-sphinx
       python3-twisted python3-zope.interface libunbound-dev libunwind-dev
13 ./configure --with-linux=/lib/modules/4.9.0-6-rpi/build
14 make
15 make install
```

Listado 4.1: Líneas de comando para la instalación de Open vSwitch 2.13.4 en RPi

1. Actualizar los repositorios y paquetes del sistema operativo, se realiza mediante las líneas 1 y 2 del Listado 4.1
2. Ingresar como superusuario para instalar con los permisos suficientes sobre los directorios, lo permite la línea 3 del Listado 4.1.
3. Descargar el archivo comprimido que contiene el instalador de Open vSwitch en su versión más estable, 2.13.4, se logra mediante la línea 4 del Listado 4.1.

4. Descomprimir el archivo instalador, se debe ejecutar la línea 5 del Listado 4.1.
5. Instalar dependencias principales que se requieren a priori para efectuar la instalación, éstas dependencias y su forma de instalar se presentan en la línea 6 del Listado 4.1.
6. De manera opcional, mediante la línea 7 del Listado 4.1, se puede verificar la lista de cabeceras de kernel disponibles. Sin embargo, por experiencia se conoce que RPi trae cabeceras por defecto, las mismas que se instalan en el paso siguiente. En la Figura 4.8 se pueden observar las cabeceras de kernel disponibles y se marcan las recomendadas a instalar en el paso posterior.

```
root@raspberrypi:/home/pi# apt-cache search linux-headers
aufs-dkms - DKMS files to build and install aufs
linux-headers-4.18.0-3-common - Common header files for Linux 4.18.0-3
linux-headers-4.18.0-3-common-rt - Common header files for Linux 4.18.0-3-rt
linux-headers-4.9.0-6-all - All header files for Linux 4.9 (meta-package)
linux-headers-4.9.0-6-all-armhf - All header files for Linux 4.9 (meta-package)
linux-headers-4.9.0-6-common - Common header files for Linux 4.9.0-6
linux-headers-4.9.0-6-common-rt - Common header files for Linux 4.9.0-6-rt
linux-headers-4.9.0-6-rpi - Header files for Linux 4.9.0-6-rpi
linux-headers-4.9.0-6-rpi2 - Header files for Linux 4.9.0-6-rpi2
linux-headers-rpi - Header files for Linux rpi configuration (meta-package)
linux-headers-rpi-rpfv - This metapackage will pull in the headers for the raspbian kernel for the
linux-headers-rpi2 - Header files for Linux rpi2 configuration (meta-package)
linux-headers-rpi2-rpfv - This metapackage will pull in the headers for the raspbian kernel for the
linux-libc-dev-alpha-cross - Linux Kernel Headers for development (for cross-compiling)
linux-libc-dev-amd64-cross - Linux Kernel Headers for development (for cross-compiling)
```

Figura 4.8: Cabeceras de kernel disponibles para RPi

7. Habiendo consultado las cabeceras, se deben instalar las correspondiente a RPi, lo cual se efectúa mediante la línea 8 del Listado 4.1.
8. Se ingresa al directorio del instalador mediante la línea 9 del Listado 4.1.
9. Se ejecuta el archivo de arranque de la instalación, tal como se muestra en la línea 10 del Listado 4.1.
10. Se revisan dependencias adicionales que requiere la instalación, esto mediante la línea 11 del Listado 4.1. El resultado mostrará paquetes que necesitan instalar o actualizar.
11. Teniendo enlistadas las dependencias faltantes, se las debe instalar; esto se ejemplifica mediante la línea 12 del Listado 4.1.
12. Se ejecuta el archivo de configuración de la instalación mediante la línea 13 del Listado 4.1, en esta línea de comando se debe indicar el módulo de kernel que se empleará para la compilación, en este caso, Open vSwitch está diseñado (sus archivos de instalación) para ejecutarse con el kernel 4.9.
13. Luego de la configuración de la compilación, se procede con la misma a través del comando `make`, indicado en la línea 14 del Listado 4.1.
14. Finalmente se ejecuta la instalación mediante `make install` indicado también, en la línea 15 del Listado 4.1

4.3.2. Configuración de Open vSwitch sobre Raspberry Pi

Al finalizar los pasos descritos para las líneas del Listado 4.1, se procede a la configuración de Open vSwitch dentro de RPi para convertir el mini ordenador en un *switch* OpenFlow, estos pasos se describen a continuación y las líneas a ejecutarse se anotan en el Listado 4.2.

```
1 cd datapath/linux
```

```
2  modprobe openvswitch
3  cat /etc/modules
4  echo "openvswitch" >> /etc/modules
5  cd ../..
6  touch /usr/local/etc/ovs-vswitchd.conf
7  mkdir -p /usr/local/etc/openvswitch
8  ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/vswitch.
       ovsschema
9  export PATH=$PATH:/usr/local/share/openvswitch/scripts
10 ovs-ctl start
11 nano sw_start.sh
12 chmod +x sw_start.sh
13 ./sw_start.sh
14 nano -e /etc/rc.local
15 /bin/sh /home/pi/openvswitch-2.13.4/sw_start.sh
16 reboot
17 ps -e | grep ovs
```

Listado 4.2: Lineas de comando para la configuración de Open vSwitch 2.13.4 en RPi

1. Dentro del directorio `openvswitch-2.13.4`, se ingresa a `datapath/linux` y se carga el módulo `openswitch` al kernel, esto se logra mediante la línea 1 y 2 del Listado 4.2.
2. Adicionalmente se agrega el módulo `openvswitch` al archivo `/etc/modules` mediante la línea 4 del Listado 4.2. La línea 3 del mismo listado permite comprobar si el módulo existe antes y después de la ejecución de la línea 4.
3. Se regresa al directorio `openvswitch-2.13.4` mediante la línea 5 del Listado 4.2.
4. Desde el directorio principal, se crea el archivo de configuración del demonio de Open vSwitch ejecutando la línea 6 del Listado 4.2.
5. Se crea un directorio para Open vSwitch desde donde se gestionan las configuraciones y bases de datos que requiere, esto mediante la línea 7 del Listado 4.2.
6. Luego de creado el directorio, se continúa con la creación de la base de datos que controla el demonio de Open vSwitch con el esquema predefinido, se ejecuta la línea 8 del Listado 4.2, mayor información se pueden encontrar en la documentación de `ovsdb-tool`[1]
7. Ejecutando la línea 9 del Listado 4.2 se define y exporta la variable global `PATH` que apunta al directorio de comandos de Open vSwitch, esto permite ejecutar, luego, la línea 10 que inicia el controlador del demonio de Open vSwitch. Este paso y algunos previos se visualizan en la Figura 4.9.
8. Mediante el editor de texto `nano` (o cualquier otro como `vi` o `gedit`), como indica la línea 11 del Listado 4.2, se crea un archivo con nombre personalizado, por ejemplo `sw_start.sh`. En este archivo de debe cargar la configuración que inicializa la base de datos del servidor de Open vSwitch, el contenido a copiar o transcribir se muestra en el Listado 4.3.

```
1 ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
2              --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
3              --private-key=db:Open_vSwitch,SSL,private_key \
```

[1]Documentación de Open vSwitch (ovsdb-tool): http://www.openvswitch.org/support/dist-docs/ovsdb-tool.1.txt

```
root@s1:/home/pi/openvswitch-2.13.4# touch /usr/local/etc/ovs-vswitchd.conf
root@s1:/home/pi/openvswitch-2.13.4# mkdir -p /usr/local/etc/openvswitch
root@s1:/home/pi/openvswitch-2.13.4# ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/vs
witch.ovsschema
root@s1:/home/pi/openvswitch-2.13.4# export PATH=$PATH:/usr/local/share/openvswitch/scripts
root@s1:/home/pi/openvswitch-2.13.4# ovs-ctl start
[ ok ] Starting ovsdb-server.
[FAIL] system ID not configured, please use --system-id ... failed!
[ ok ] Configuring Open vSwitch system IDs.
[ ok ] Starting ovs-vswitchd.
[ ok ] Enabling remote OVSDB managers.
root@s1:/home/pi/openvswitch-2.13.4#
```

Figura 4.9: Inicialización del controlador del demonio de Open vSwitch

```
4             --certificate=db:Open_vSwitch,SSL,certificate \
5             --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
6             --pidfile -detach
7 ovs-vsctl --no-wait init
8 ovs-vswitchd --pidfile --detach
9 ovs-vsctl show
```

Listado 4.3: Archivo sw_start.sh

9. Después, se otorga permisos de ejecución mediante la línea 12 del Listado 4.2, para luego ejecutar el archivo empleando la línea 13 del mismo listado. Esto inicializa el demonio de Open vSwitch con las configuraciones, bases de datos y servicios necesarios para la funcionalidad del *switch*. Este paso y algunos otros previos se pueden ver al inicio de la Figura 4.11.
10. Para que esta ejecución recursiva sea automática se debe editar el archivo `rc.local` y se debe agregar la línea 15 en este archivo, antes de la línea `exit 0`, como se observa en la Figura 4.10; luego guardar los cambios y salir.

```
  [1/1]                          /etc/rc.local                          Modificado


# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

/bin/sh /home/pi/openvswitch-2.13.4/sw_start.sh

exit 0



^G Ver ayuda  ^O Guardar    ^W Buscar      ^K Cortar txt  ^J Justificar  ^C Posición    M-U Deshacer
^X Salir      ^R Leer fich. ^\ Reemplazar  ^U Pegar txt   ^T Ortografía  ^_ Ir a línea  M-E Rehacer
```

Figura 4.10: Configuración en el archivo `rc.local`

11. Para que los cambios en el archivo `rc.local` surtan efecto, se debe reiniciar mediante `reboot` y cuando inicie nuevamente se debe ingresar a una terminal de la RPi (por SSH o directamente en la RPi) y ejecutar el comando de la línea 17 del Listado 4.2. Esto permite evidenciar que los servicios están ejecutándose. Así se muestra al final de la Figura 4.11.

```
root@s1:/home/pi/openvswitch-2.13.4# nano sw_start.sh
root@s1:/home/pi/openvswitch-2.13.4# chmod +x sw_start.sh
root@s1:/home/pi/openvswitch-2.13.4# ./sw_start.sh
ovsdb-server: /usr/local/var/run/openvswitch/ovsdb-server.pid: already running as pid 1990, aborting
ovs-vswitchd: /usr/local/var/run/openvswitch/ovs-vswitchd.pid: already running as pid 2065, aborting
78aafe75-37e0-46e5-a63a-426fee7d84eb
    ovs_version: "2.13.4"
root@s1:/home/pi/openvswitch-2.13.4# nano -e /etc/rc.local
root@s1:/home/pi/openvswitch-2.13.4# ps -e | grep ovs
 1990 ?        00:00:00 ovsdb-server
 2065 ?        00:00:00 ovs-vswitchd
root@s1:/home/pi/openvswitch-2.13.4#
```

Figura 4.11: Verificación de los procesos activos de Open vSwitch

4.3.3. Configuración de *switch* OpenFlow sobre Raspberry Pi empleando Open vSwitch

Una vez instalado y configurado Open vSwitch dentro de Raspberry Pi, se procede a la configuración del *switch*, para tal fin se recomienda crear un *script* personalizado para cada *switch*. En este apartado se muestra un ejemplo para la configuración del *switch* Kappa. Se muestra el contenido del *script* `br0.sh` en el Listado 4.4, en el cual se ha configurado el *switch* Kappa con `id=0000000000000002` y con 4 puertos Ethernet. Los adaptadores USB-Ethernet deben estar conectados para su correcta configuración, los mismos que se numeran en el orden de conexión.

El *script* contiene configuraciones sobre una interfaz tipo puente (*brigde*) en la que se agrega los puertos Ethernet y se configura características de Open vSwitch sobre el *switch* y el controlador al cual se conectará. Cada configuración se encuentra comentada, como se observa en el Listado 4.4.

Para ejecutar el *script* `br0.sh` se debe ingresar al modo superusuario mediante `sudo su`, antes de ello debe asegurar el permiso de ejecución con `chmod +x br0.sh`. Desde la terminal en modo superusuario, la ejecución es simple con `./br0.sh`.

```
1  ifconfig eth0 down #Se inhabilitan las interfaces Ethernet
2  ifconfig eth1 down
3  ifconfig eth2 dowm
4  ifconfig eth3 down
5  ovs-vsctl --if-exists del-br br0 #Borra interfaz br0 en caso de existir
6  ovs-vsctl add-br br0 #Agrega interfaz br0
7  ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000002 #
       Configura id del switch
8  ovs-vsctl add-port br0 eth0 #Agrega la interfaz eth0 al switch
9  ovs-vsctl set interface eth0 ofport_request=1 #Configura eth0 como Puerto 1
10 ovs-vsctl add-port br0 eth1 #Agrega la interfaz eth1 al switch
11 ovs-vsctl set interface eth1 ofport_request=2 #Configura eth1 como Puerto 2
12 ovs-vsctl add-port br0 eth2 #Agrega la interfaz eth2 al switch
13 ovs-vsctl set interface eth2 ofport_request=3 #Configura eth2 como Puerto 3
14 ovs-vsctl add-port br0 eth3 #Agrega la interfaz eth3 al switch
15 ovs-vsctl set interface eth3 ofport_request=4 #Configura eth3 como Puerto 4
16 ifconfig eth0 0 up #Se habilitan las interfaces Ethernet
17 ifconfig eth1 0 up
18 ifconfig eth2 0 up
19 ifconfig eth3 0 up
```

```
20 ovs-vsctl set-controller br0 tcp:192.168.1.166:6653 #Configura conexion TCP
       hacia el controlador
21 ovs-vsctl set controller br0 connection-mode=out-of-band #Configura la
      conexión con el controlador en modo fuera de banda
22 ovs-vsctl set bridge br0 protocols=OpenFlow13 #Predetermina el uso de
      OpenFlow 1.3
23 ovs-vsctl set-fail-mode br0 secure #Configura modo seguro ante fallos
24 ovs-vsctl get Bridge br0 datapath-id #Muestra el identificador del switch
25 ovs-vsctl show #Muestra resumen de las configuraciones cargadas
26 ovs-vsctl -- --columns=name,ofport list Interface #Lista los puertos y su
      identificador
```

Listado 4.4: Configuración de *switch* en RPi

CAPÍTULO 5

Pruebas de funcionamiento y análisis de resultados

En este capítulo se describen los resultados obtenidos en las pruebas de funcionamiento, considerando cada práctica del entorno de emulación, también se anotan y analizan los resultado de funcionamiento de la implementación física.

Cabe indicar que cada guía desarrollada en el Apéndice A cuenta con la sección de **Verificación de funcionamiento y resultados esperados**, lo que corrobora el funcionamiento en el entorno de emulación y ahí se detalla más específicamente los resultados obtenidos. A continuación se resumen los resultados más relevantes.

5.1. Resultados obtenidos en el entorno de emulación

Las prácticas presentadas en la Sección 4.1 fueron desarrolladas para evaluar los controladores y las aplicaciones disponibles en referencias bibliográficas, con la finalidad de dar una variante o modificación que permita ir un paso adelante cubriendo aspectos que se cree deben poseer las redes definidas por software.

A continuación se describen los resultados obtenidos en cada práctica en base a las instrucciones y funcionamiento indicados en el Apéndice A.

5.1.1. Práctica 1

Al finalizar la Práctica 1 se logró conectar el plano de datos con el plano de control, para el plano de control se consideraron los controladores POX, Ryu y ODL. El desarrollo de la práctica proporciona una idea clara de qué controlador usar en el entorno de pruebas, no específicamente por aspectos de conexión con el plano de datos sino también considerando el lenguaje de programación usado. Es decir, se considera la facilidad de desarrollo de aplicaciones que se evidencia en cada uno. En ese contexto, subjetivamente se elige al controlador Ryu para ser implementado en la mayoría de escenarios de prueba, pues permite la escritura de aplicaciones en Python que es uno de los lenguajes más conocidos y empleados debido a su simplicidad en la lógica de programación, a más de que cuenta con gran cantidad de librerías, documentación y ejemplos.

En la Figura 5.1 se evidencia la ejecución de una topología *Minimal* de Mininet y la comunicación de conmutador `s1` con el controlador POX, así como se revisan su tabla de entradas de flujo.

Figura 5.1: Conexión entre `s1` y el controlador SDN POX.

Para el controlador Ryu, de manera similar que para POX, se evidenció la conexión del controlador con la topología y se consultan la tabla de entradas de flujo en el conmutador `s1`, lo muestra la Figura 5.2

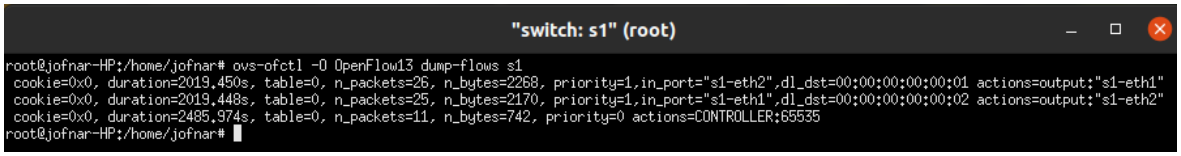


Figura 5.2: Entradas de flujo en *s1*

En el caso del controlador ODL, gracias a su interfaz web OpenDayLigth User eXperience (DLUX), se puede evidenciar la Figura 5.3 donde se presentan las estadísticas del conmutador `s1` que permite constatar la comunicación del plano de datos con el plano de control.

Node Connector Statistics for Node Id - openflow:1

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:1:2	26	2720	1916	233128	0	0	0	0	0	0	0	0
openflow:1:1	26	2720	1916	233128	0	0	0	0	0	0	0	0
openflow:1:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0
openflow:1:3	26	2722	1916	233212	0	0	0	0	0	0	0	0

Figura 5.3: Estadísticas en el nodo *s1*

Otros resultados evidenciados de esta práctica así como mayores detalles y análisis de los mismos,

se pueden revisar en el Apéndice A, específicamente en A.1.5.

5.1.2. Práctica 2

El resultado tras desarrollar la Práctica 2 es la familiarización de los comandos principales en el emulador de Mininet y la destreza adquirida para crear una topología personalizada usando el lenguaje de Python, de tal manera que se obtuvieron los conocimientos suficientes para abordar, diseñar e implementar las topologías necesarias en prácticas posteriores.

En la Figura 5.4 se observa la ejecución de una topología *Tree* (predefinida en Mininet) que se conecta a un controlador Ryu.

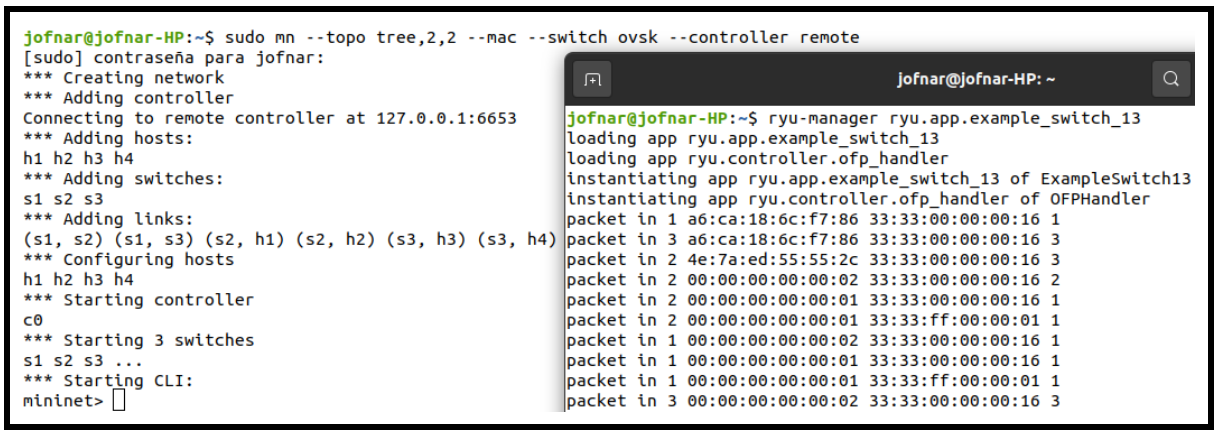


Figura 5.4: Resultado de ejecutar la topología *Tree* y conexión con el controlador Ryu.

La evaluación de principales comandos en Mininet da como resultado salidas de información o acciones sobre la topología, algunas de las salidas se muestran a continuación. La Figura 5.5 muestra el caso de `net`, la Figura 5.6 presenta la salida de `dump` y la Figura 5.7 evidencia la ejecución de un ensayo con iPerf. La visualización de estas salidas permite una mejor comprensión del entorno de Mininet.

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
c0
```

Figura 5.5: Resultado de ejecutar el comando `net` en el terminal de Mininet.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=20189>
<Host h2: h2-eth0:10.0.0.2 pid=20191>
<Host h3: h3-eth0:10.0.0.3 pid=20193>
<Host h4: h4-eth0:10.0.0.4 pid=20195>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=20200>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=20203>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=20206>
<RemoteController c0: 127.0.0.1:6653 pid=20183>
```

Figura 5.6: Resultado de ejecutar el comando `dump` en el terminal de Mininet.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['32.1 Gbits/sec', '32.1 Gbits/sec']
```

Figura 5.7: Resultado de ejecutar el comando `iperf` en el terminal de Mininet.

Por otra parte, la guía propone una topología personalizada, la misma que se desarrolla en un *script* Python para su ejecución en Mininet, es así que la creación de los elementos de red y sus enlaces se verifican a través de comando `net`, tal salida se indica en la Figura 5.8.

```
mininet> net
h1 h1-eth1:s2-eth2
h2 h2-eth1:s3-eth2
h3 h3-eth1:s4-eth2
h4 h4-eth1:s5-eth2
s1 lo:  s1-eth1:s5-eth1 s1-eth2:s2-eth1 s1-eth3:s3-eth1 s1-eth4:s4-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:h1-eth1
s3 lo:  s3-eth1:s1-eth3 s3-eth2:h2-eth1
s4 lo:  s4-eth1:s1-eth4 s4-eth2:h3-eth1
s5 lo:  s5-eth1:s1-eth1 s5-eth2:h4-eth1
c0
```

Figura 5.8: Resultado de ejecutar el comando *net* en el terminal de Mininet.

La topología personalizada e implementada en Mininet se conecta a ODL para aprovechar la interfaz gráfica DLUX y visualizar la topología, tal como se muestra en la Figura 5.9, esta topología corresponde a la planteada inicialmente en la guía práctica.

Los resultados anotados validan la creación de una topología predefinida y una personalizada, además de su conexión con distintos controlares, así como se presenta la familiarización con comandos de Mininet. Otros resultados de esta práctica así como mayores detalles y análisis de los mismos, se pueden revisar en el Apéndice A, específicamente en A.2.5.

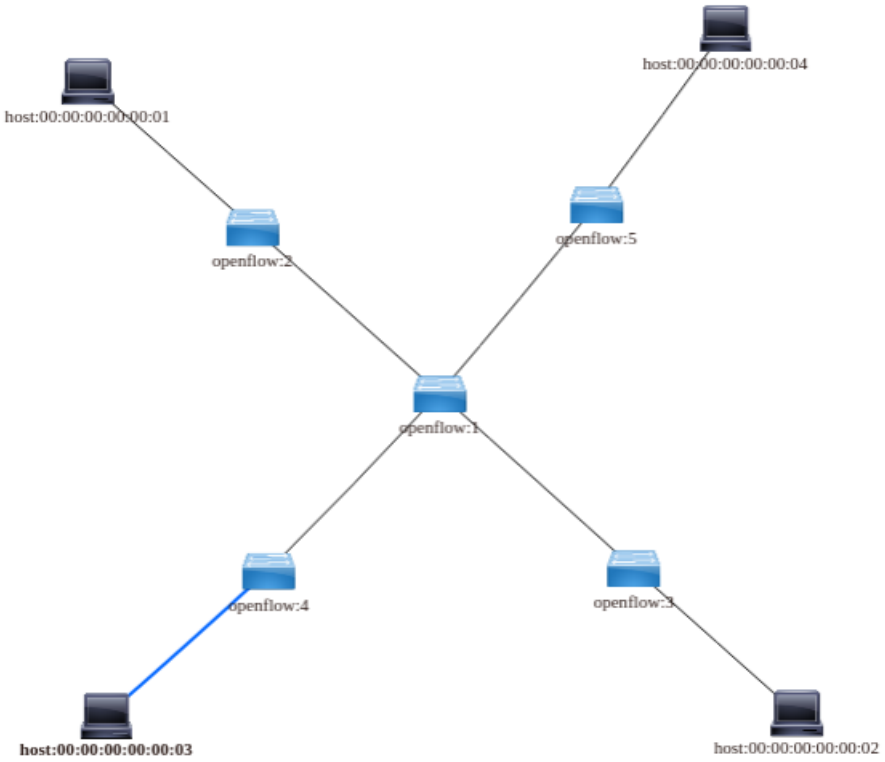


Figura 5.9: Visualización de la topología implementada desde la interfaz del controlador ODL.

5.1.3. Práctica 3

La evaluación de la funcionalidad de "Agregación de enlace" ha permitido configurar interfaces de enlaces así como "Grupos de agregación de enlace" en una topología de mayor complejidad que la presentada en la documentación de referencia. Las pruebas evidenciaron más claramente el comportamiento de la distribución de tráfico en enlaces dedicados para tráfico ascendente, descendente y bidireccional, por igual se corrobora la tolerancia a fallos. Además se comprueba la escalabilidad que puede soportar una aplicación con una leve modificación para emplearla en una topología distinta al ejemplo documentado.

A breves rasgos, en la Figura 5.10 se puede observar la distribución de tráfico que se logra en la topología planteada en la práctica (de mayor complejidad en comparación al ejemplo documentado). Esto, gracias a la creación de los dos grupos de enlace.

Por otra parte, la puesta a prueba de la tolerancia a fallos resulta en la detección de la caída del enlace y la inhabilitación del mismo bajo el protocolo LACP, esto se muestra en la Figura 5.11

Durante la verificación de funcionamiento, se van evidenciando las entradas que se crean y se eliminan en las tablas de flujo, éstos y otros resultados de esta práctica así como mayores detalles y análisis de los mismos, se pueden revisar en el Apéndice A, específicamente en A.3.5.

5.1.4. Práctica 4

En la Práctica 4 se aplicaron tablas de grupo para la división porcentual de tráfico, esta práctica estuvo basada en una aplicación ya desarrollada con anterioridad sobre el manejo de tablas de grupo.

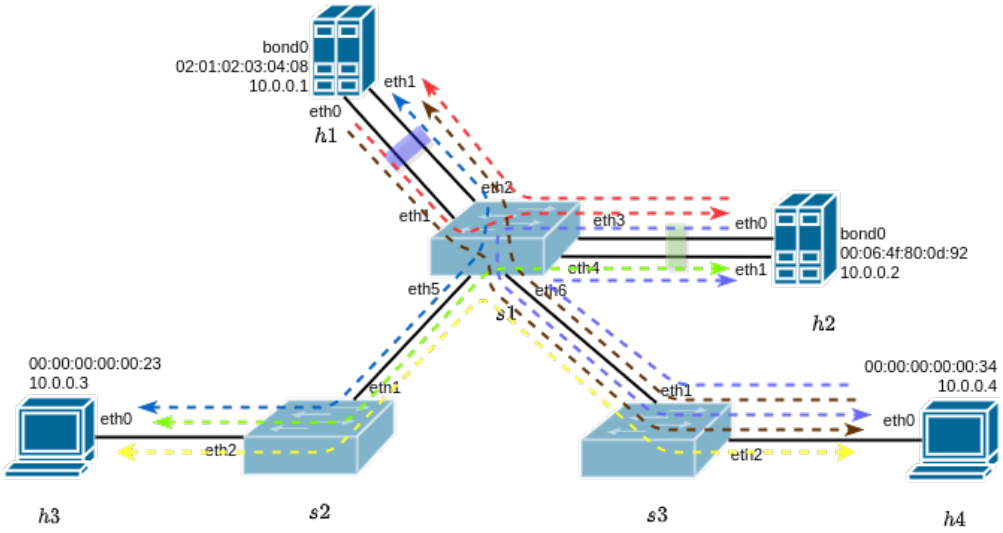


Figura 5.10: Distribución de Tráfico a través de los Grupos de Enlace

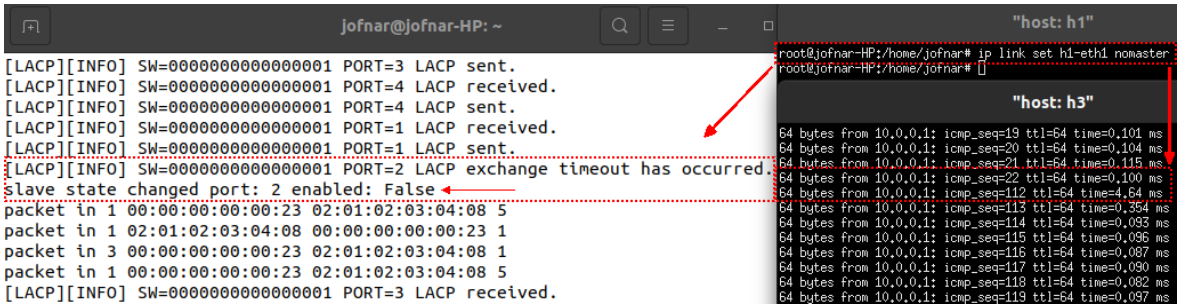


Figura 5.11: Separación de la interfaz eth1 de h1 y comportamiento esperado

Para su desarrollo se dio un enfoque diferente en donde se aplicaron tablas de grupo considerando el tipo de grupo `SELECT` de tal manera que se definen 2 rutas. El diseño contemplaba que ingresa el 100 % de tráfico y las acciones de grupo en función de los pesos asignados divide el tráfico, redirigiendo por una ruta el 65 % de tráfico total y por la otra ruta el 35 %. Al finalizar la práctica, los resultados evidenciaron el 66,456 % de tráfico en una ruta y 33,544 % en la otra. Es decir, se tuvo resultados muy aproximados a lo deseado, la leve diferencia se debe al intercambio de mensajes de control y de otros protocolos que surgen de manera continua entre *switches*. La importancia de esta práctica radica en que no se colapsa una sola ruta como sucede en el enrutamiento IP clásico basado en dirección de destino (mejor camino), reduciendo así posibles pérdidas de información y procurando otorgar la QoS requerida por diferentes servicios actuales.

Tras el desarrollo de la aplicación para la división porcentual, según la topología y diseño de tráfico de la Figura 5.12, se ejecuta con el controlador Ryu y se efectúan las pruebas de funcionamiento. En la Figura 5.13 se muestra la consulta a las tablas de flujo sobre el tráfico cursado por los puertos de `s1`, esta información permite obtener los porcentajes descritos en el párrafo anterior.

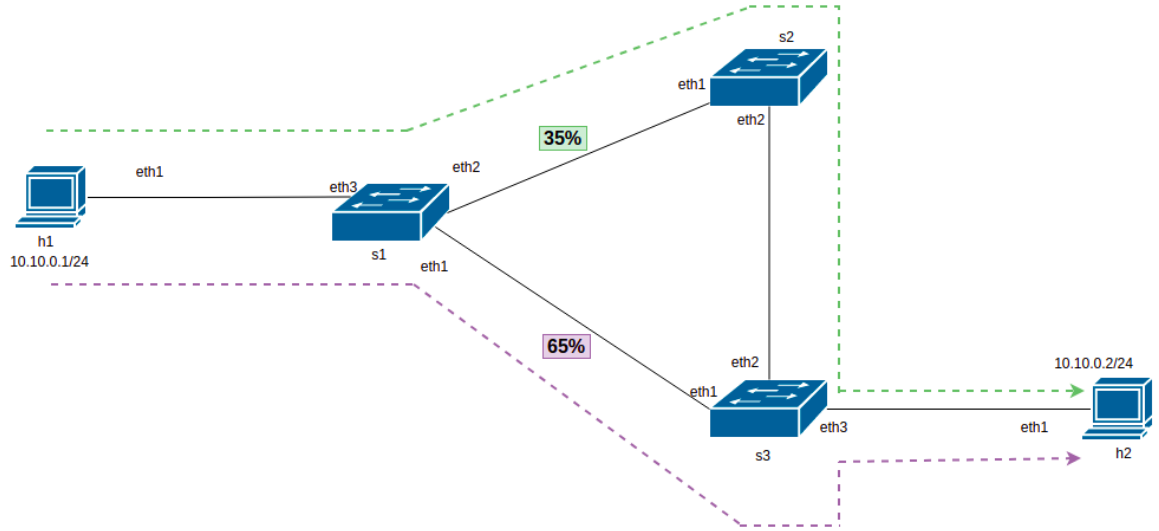


Figura 5.12: Topología para división porcentual de tráfico.

```
root@jhossPC:~/Practicas/P4# ovs-ofctl -O openflow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
  port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=74.627s
  port  "s1-eth1": rx pkts=1547535, bytes=75923773015, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=899334, bytes=59386597, drop=0, errs=0, coll=0
           duration=74.631s
  port  "s1-eth2": rx pkts=781132, bytes=39363891747, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=385175, bytes=25440555, drop=0, errs=0, coll=0
           duration=74.633s
  port  "s1-eth3": rx pkts=1284468, bytes=84820620, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=2328651, bytes=115287662285, drop=0, errs=0, coll=0
           duration=74.632s
root@jhossPC:~/Practicas/P4#
```

Figura 5.13: Verificación de división porcentual de tráfico

Durante la verificación de funcionamiento se evidencian las entradas de tablas de flujo y tablas de grupo, que se crean para permitir la división porcentual, éstos y otros resultados de esta práctica así como mayores detalles y análisis de los mismos, se pueden revisar en el Apéndice A, específicamente en A.4.5.

5.1.5. Práctica 5

Al desarrollar la Práctica 5 se obtiene como resultado un tratamiento diferenciado de tráfico dependiendo el protocolo. En el caso que el tráfico sea UDP, éste sigue la ruta `s1-s4-s3` y si el tráfico es TCP toma la ruta `s1-s2-s3`, según la topología implementada y mostrada en la Figura 5.14. Para el desarrollo de la práctica se considera la creación de "Tablas de grupo ALL", que evita la pérdida de información al momento de definir la ruta por protocolo. En aplicaciones reales, con esta práctica, se contribuye a otorgar rutas de acuerdo a la sensibilidad que presentan los servicios, ya sean estos de: *streaming*, vídeo, transferencia de archivos, entre otros. De cierta manera se aborda y se asegura

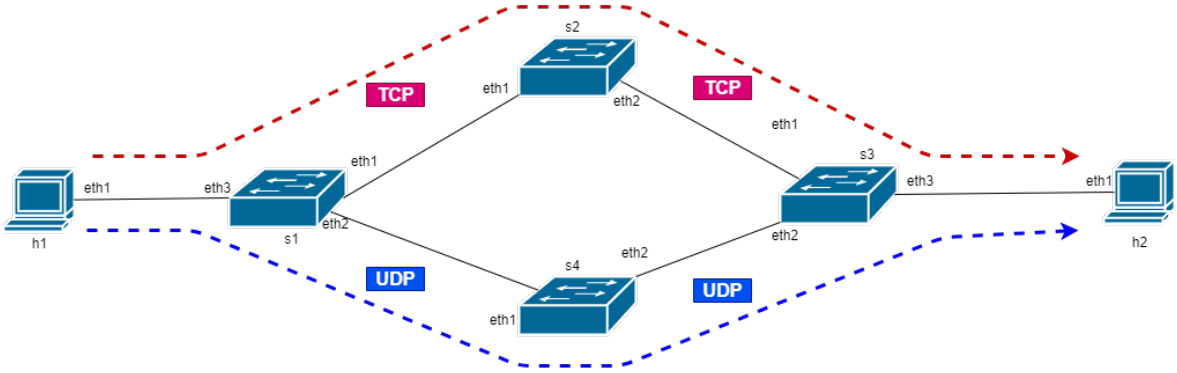


Figura 5.14: Topología para direccionamiento de tráfico.

niveles de QoS diferenciado para la red. Los tráficos prioritarios de tiempo real se direccionan por las mejoras rutas, en tanto que los tráficos de menor prioridad igualmente son transmitidos aunque por rutas menos eficientes.

Los resultados se encuentran fundamentados tras el análisis de las entradas en las tablas de flujo y tablas de grupo de cada uno de los conmutadores, éstos y otros resultados de esta práctica así como mayores detalles y análisis de los mismos, se pueden revisar en el Apéndice A, específicamente en A.5.5.

5.1.6. Práctica 6

En la Práctica 6 se propone una aplicación que permite dar un tratamiento diferenciado al tráfico. Pero, a diferencia de la práctica anterior, no se consideran tablas de grupo sino múltiples tablas de flujo (Proceso de entubado de las tablas de flujo). En esta práctica cada *switch* cuenta con 3 tablas de flujo que son las encargadas de direccionar el tráfico según las coincidencias (`match`). De principio, en el primer *switch* `s1` se da un tratamiento diferenciado en caso de que el tráfico sea TCP o UDP; en caso de que el tráfico sea TCP, éste vuelve a ser tratado en `s2` dependiendo si el tráfico es FTP o HTTP. Si el tráfico es UDP, éste se dirige hacia `s3`, el cual da un tratamiento al tráfico sea éste RTP o RTCP. Al finalizar la práctica se tienen las siguientes rutas según el protocolo.

- FTP: Ruta bidireccional `h5-s1-s2-h1`
- HTTP: Ruta bidireccional `h5-s1-s2-h2`
- RTP: Ruta bidireccional `h5-s1-s3-h3`
- RTCP: Ruta bidireccional `h5-s1-s3-h4`

Para definir las rutas según protocolo se tomó en cuenta: dirección origen, dirección destino, protocolo y puerto. Los resultados de manera especifica y con mayor detalle se presentan en A.6.5

5.1.7. Práctica 7

La evaluación de una aplicación existente para el descubrimiento de rutas, en esta práctica, permitió familiarizarse con funciones y eventos OpenFlow programados en Ryu que son utilizados en la Práctica 8. Los resultados visibles obtenidos de la práctica son un listado de 3 rutas posibles que conectan 2 *hosts* extremos, se muestran los costes asociados al camino y el tiempo necesario de la instalación del camino.

La aplicación para el controlador se encuentra documentada; sin embargo, con leves modificaciones ha permitido evaluarla en una topología diferente y más didáctica, dando lugar a propuestas que mejoren sus funcionalidades. Estas propuestas van al campo de la administración y monitoreo de estados de enlace y la ingeniería de tráfico (TE) por medio del reenrutamiento rápido, abordados en la Práctica 8. Los resultados obtenidos, de manera especifica y con mayor detalle, se presentan en A.7.5

5.1.8. Práctica 8

Al finalizar la práctica se obtiene una aplicación de reenrutamiento rápido que se evalúa sobre la misma topología de la Práctica 7. Esta topología poseía 1 ruta principal y 2 de respaldo, esto permitió la comprobación de la funcionalidad ante la caída de un enlace y luego ante la recuperación del mismo. Los resultados dan una alta expectativa al enfoque SDN pues evidencian las altas capacidades y flexibilidad que puede tener una red a través de la programabilidad.

Las pruebas de conectividad después de la caída y recuperación del enlace muestran que la conmutación de rutas es inmediata y muy abstracta al usuario o a la red puesto que no se pierde conectividad en ningún momento. Estos resultados, de manera detallada y con su respectivo análisis se muestra en A.8.5

5.1.9. Práctica 9

La Práctica 9 se basa en la aplicación `rest_firewall` proporcionada por Ryu, la cual se encuentra ejemplificada en diferentes trabajos anteriormente realizados, en esta se permite o bloquea el tráfico entre distintos *hosts*.

En base a la documentación de referencia, en este trabajo se establecen reglas de seguridad mediante API REST usando la herramienta POSTMAN, no solo se plantean reglas que permitan o bloqueen el tráfico entre diferentes *hosts*, sino que se manejan diferentes tipos de parámetros para permitir o bloquear un tráfico. Esta práctica da como resultado una red donde se permite el curso de tráfico ICMP entre `h1 y h2`, y el curso de todo tipo de tráfico entre `h1 y h3`. Posteriormente se establece una regla que bloquee el tráfico TCP en el puerto 80 que corresponde al protocolo HTTP, en `h3`. El desarrollo de esta práctica es de importancia en el momento de establecer reglas de acceso a sitios específicos, por ejemplo, en un plantel educativo se puede restringir el acceso a redes sociales, siendo aplicable para un *proxy*.

Los resultados con mayores descripciones y detalles, se encuentran documentados en A.9.5.

5.1.10. Práctica 10

En la Práctica 10 se obtuvo como resultado un *Firewall* simple, es decir, uno que permite o bloquea el tráfico entre dos hosts. También se creó un usuario `usrtesis` al cual se le otorga los permisos necesarios para que acceda al controlador ODL como administrador. Es decir, pueda realizar operaciones que afecten a la red. Para el desarrollo de la práctica se tomó como base la documentación proporcionada por el sitio oficial de OpenDayligth (ODL) acerca de los módulos NIC y la aplicación de autenticación, autorización y contabilidad (Authentication, Authorization and Accounting (AAA)). Esta práctica aborda un enfoque de seguridad y el uso del controlador ODL para prácticas posteriores. Es así que los resultados a detalle se describen en A.10.5

5.1.11. Práctica 11

Al finalizar la Práctica 11 se logra interconectar una red tradicional a una red SDN mediante el protocolo BGP. La red tradicional se compone de 3 sistemas autónomos que incluyen enrutadores, *switches* y *hosts*, mientras que en la red SDN se tiene un único sistema autónomo. Para el desarrollo de esta práctica se tomó como base la documentación proporcionada por el sitio oficial de OpenDayLigth acerca del complemento BGP. Los resultados se desglosan en A.11.5.

5.1.12. Práctica 12

La Práctica 12 se basa en la documentación proporcionada por la página oficial de Open vSwitch (OVS) para la configuración de un túnel VXLAN, al finalizar la práctica se logra conectar dos servidores y sus redes dominio, ubicados en diferentes lugares (simulados en máquinas virtuales). Utilizando un túnel VXLAN, dichos servidores están conectados a un controlador Ryu externo, esto se muestra en detalle en A.12.5.

5.2. Resultados obtenidos en el entorno de implementación física

El diseño planteado y considerado en la Sección 4.2 se ha materializado resultando en un laboratorio de pruebas para SDN de menor escala. La estructura de soporte diseñada ha permitido colocar los conmutadores (Raspberry Pi) con los adaptadores USB-Ethernet que actúan como puertos Ethernet de los *switches* OpenFlow, esto se evidencia en la Figura 5.15. Aquí se ha etiquetado los *switches* y sus puertos para facilitar la identificación y conexiones en las implementaciones.

El entorno de implementación física cumple con su objetivo, facilitar la evaluación de aplicaciones desarrolladas en el entorno de emulación. Como resultado se obtiene material multimedia recopilado en vídeos, donde se explica los pasos para la implementación y la verificación de resultados.

Este material multimedia se ha almacenado en una ubicación de nube, el acceso será permanente mientras la cuenta institucional hospedadora tenga vigencia. El acceso está permitido para usuarios con credenciales del correo institucional de la Universidad de Cuenca mediante el siguiente enlace:

Laboratorio RPi SDN

A continuación se muestra un registro fotográfico de uno de los vídeos referente a una de las prácticas implementadas en el laboratorio. En este caso se presentan capturas de la implementación física de la Práctica 5: Tratamiento diferenciado de tráfico TCP y UDP.

En la Figura 5.16 se muestra la ejecución del controlador SDN Ryu, con la aplicación correspondiente `(tcp_udp.py)` sobre un ordenador. Para la implementación física de la Práctica 5 se requirió 3 ordenadores, de los cuales 2 actúan como puntos finales o terminales y uno como controlador.

Figura 5.15: Laboratorio para SDN de menor escala con Raspberry Pi

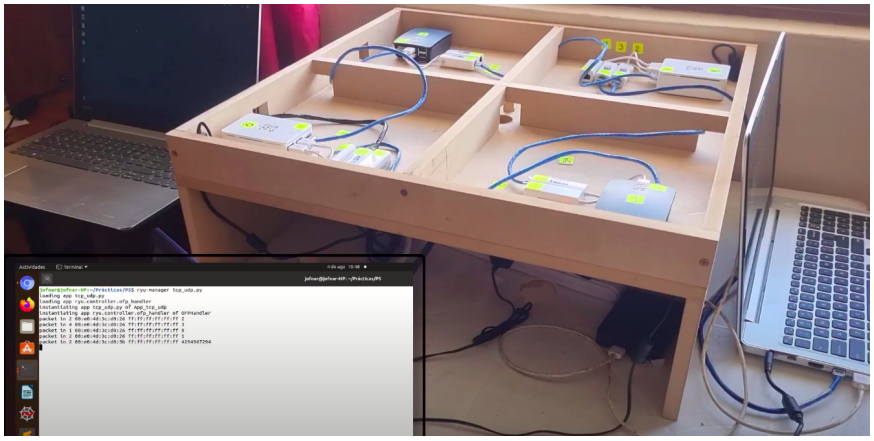


Figura 5.16: Ejecución de Controlador, conectado por WiFi al plano de datos

Con el controlador en funcionamiento se procede a realizar pruebas del correcto funcionamiento de la aplicación desarrollada. En primer lugar se usa la herramienta iPerf para generar tráfico UDP y verificar las entradas de flujo generadas en los *switches* respectivos. Se observan entradas de flujo en `s1, s4, s3` lo que corrobora que el camino que cursa el tráfico UDP es `s1-s4-s3`. Esto se aprecia en

la Figura 5.17.

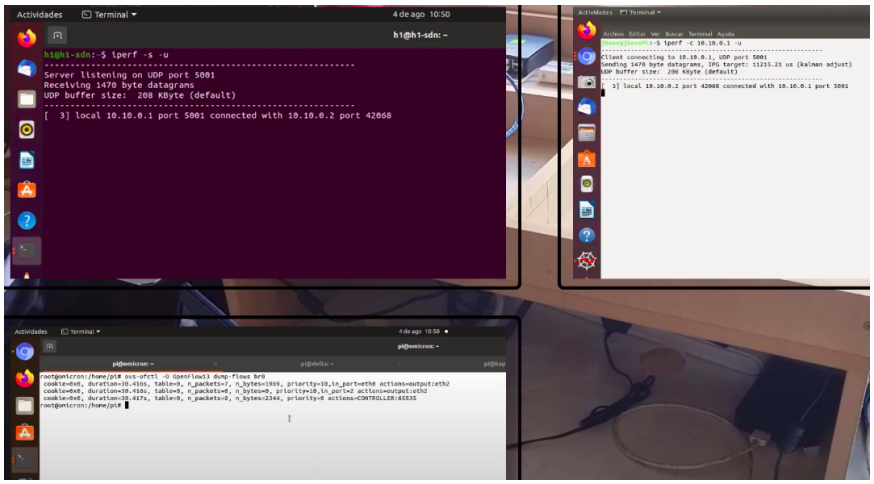


Figura 5.17: Pruebas a través de iPerf

En el entorno de implementación física además de realizar pruebas de funcionamiento mediante iPerf, se realizan pruebas utilizando el reproductor de vídeo VLC que permite tráfico UDP y TCP simultaneo empleando vídeo *streaming* con RTSP. Esto se muestra en la Figura 5.18.

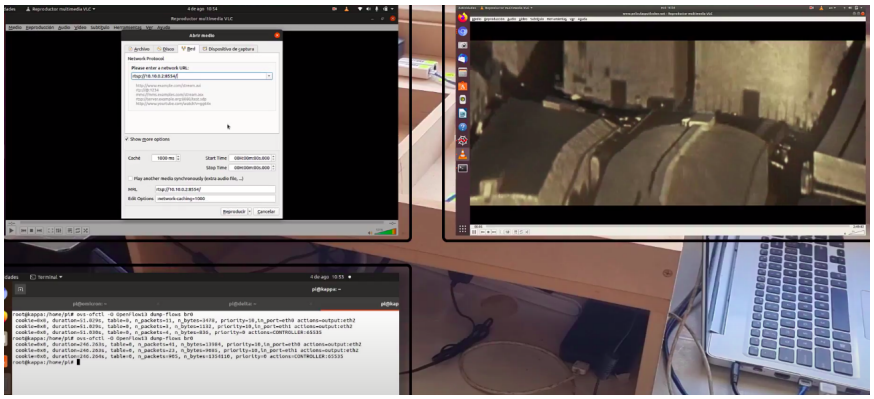


Figura 5.18: Configuraciones para pruebas de tráfico UDP-TCP por vídeo *streaming* con RTSP

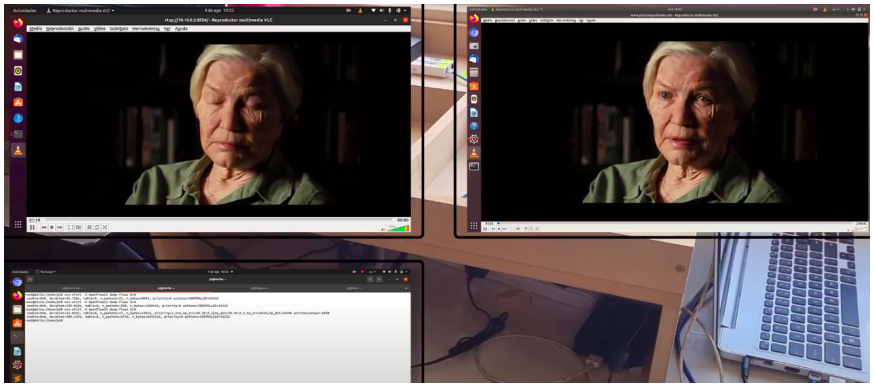


Figura 5.19: Emisión y demanda de vídeo desde los *hosts*

Al realizar las pruebas mediante VLC, utilizando *streaming* bajo RTSP, se procede a revisar las entradas de las tablas de flujo en los *switches* `s2` y `s4`. Estos flujos corresponden a ambos tipos de tráfico, algunos a UDP y otros a TCP, los mismos que se instalan debido a que RTSP emplea una conexión TCP para el control y otras dos conexiones UDP para el envío de datos y reenvió de peticiones. Es así que la conexión de control usa la ruta `s1-s2-s3` mientras que los datos y el reenvío cursan a través de `s1-s4-s3`.

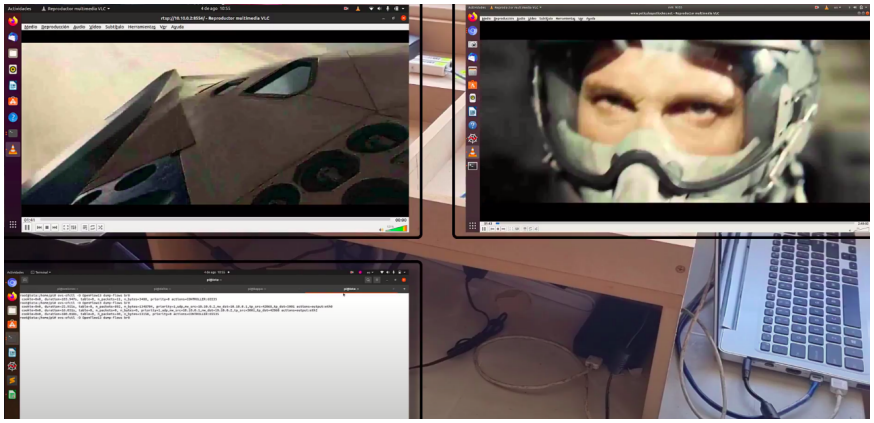


Figura 5.20: Retraso en transmisión.

Dado que se trabaja en un entorno realista, sujeto a pérdidas y a un rendimiento limitado por parte de los *switches*, se presenta un retraso de tiempo en la recepción de la transmisión, esto se puede apreciar en la Figura 5.20, así como también en la Figura 5.19

Este laboratorio de implementación física permite ejecutar algunas de las prácticas planteadas y desarrollas en el entorno de emulación, donde el procedimiento para su implementación requiere de la configuración de los *switches* en cuanto a sus identificadores y puertos, así también, se deben realizar las conexiones físicas según sea la topología. Los pasos para su ejecución y verificación de funcionamiento son los mismos que constan en las guías respectivas. De manera adicional se pueden emplear recursos como VLC, Wireshark u otros que permiten analizar más a detalle el comportamiento de la SDN en cada uno de los casos de estudio.

CAPÍTULO 6

Conclusiones y Recomendaciones

En este capítulo se presentan las conclusiones del trabajo realizado, y las recomendaciones resultantes del diseño del entorno de análisis y prueba.

6.1. Conclusiones

El trabajo realizado es el diseño e implementación de un entorno de pruebas de análisis, experimentación, evaluación y desarrollo de redes definidas por software. El entorno se compone de 12 prácticas que permiten la realización y evaluación de distintas aplicaciones SDN. Estas prácticas han sido emuladas usando Mininet con controladores remotos (Ryu, ODL y POX) con la finalidad de aportar en el estudio, investigación y desarrollo futuro de SDN en la Universidad Cuenca.

Al finalizar este trabajo se ha adquirido conocimiento importante acerca de las Redes definidas por software y los beneficios que esta posee frente a las redes tradicionales. Se ha visto las ventajas de la separación del plano de datos del plano de control y como esto proporciona herramientas, mediante la programabilidad, para afrontar desafíos en el despliegue de redes modernas.

El problema principal en el desarrollo de este trabajo fue la variabilidad en las versiones del *software* utilizado, produciendo incompatibilidad y perdiendo abstracción. En el caso del controlador ODL varias funcionalidades eran dadas de baja en una versión superior del controlador, esto conllevó a la utilización de diferentes versiones de dicho controlador según la aplicación a desarrollar; este conflicto se presenta por la falta de continuidad de los proyectos de código abierto, algunos se abandonan por que son abordados en iniciativas privadas.

En este trabajo se ha plasmado una ínfima parte de todo lo que se puede lograr con SDN; se ha hecho hincapié en aplicaciones referentes a *Firewall*, ingeniería de tráfico TE, seguridad, administración de controlador, monitoreo, entre otros. Al utilizar Mininet como entorno de emulación se ha facilitado la creación y configuración de topologías y su estudio a través de pruebas con controladores remotos. SDN.

Además, el estudio de aplicaciones existentes en los campos mencionados en el párrafo anterior, han permitido proponer algunas variantes. En este aspecto, entre los más significativos aportes se pueden

anotar:

- Tratamiento diferenciado de flujos por protocolo (Práctica 5)
- Clasificación de tráfico por entubado de tablas de flujo (Práctica 6)
- Reenrutamiento rápido por descubrimiento de rutas (Práctica 8)
- Diversificación de reglas para *Firewall* (Práctica 9)

Para la mayoría de aplicaciones se empleó el controlador SDN Ryu, el cual está basado en el lenguaje de programación Python, lo cual permite un ambiente más amigable para el desarrollo de aplicaciones.

El entorno de pruebas en emulador partió del diseño e implementación de prácticas a desarrollar, cada una con su estructura para un estudio válido y ordenado. Las pruebas y resultados exitosos han permitido saltar del entorno de emulación al entorno de implementación física.

El uso de Raspberry Pi ha facilitado crear un laboratorio SDN de menor escala para evaluar aplicaciones que han pasado por un entorno de emulación. Esto evidencia el potencial de las SDN que pueden ser implementadas y escalables sin las limitaciones de compatibilidad por el fabricante o desarrollador. El uso de aplicaciones y *software* de código abierto permite evitar implicaciones de altos costes, dado que se puede implementar o adaptar a *hardware* de bajo coste, otorgando un amplio soporte a SDN.

Este trabajo ha desarrollado todas las implementaciones prácticas procurando el uso de las últimas versiones de *software*, estables y disponibles, tanto en los escenarios de emulación como en laboratorio físico. Escasos trabajos relacionados usan versiones actuales en controladores, *switch* virtual y *software* en general.

En el contexto anterior, el kernel actual de Linux (5 o superior), portado por Raspberry Pi en su sistema operativo Raspberry Pi OS, presenta ciertas incompatibilidades con la implementación de OVS para lograr convertirlo en *switch*. Muchos trabajos relacionados evaden la incompatibilidad bajando la versión del kernel, es decir, trabajan con versiones antiguas de Raspbian (predecesor de Raspberry Pi OS); sin embargo, el presente trabajo ha logrado compatibilizar el kernel 5 y OVS sin recurrir a evadir el sistema operativo actual. Situaciones similares se presentaron en cierto *software* como Ryu y Mininet con las versiones de Python.

Al finalizar este trabajo se puede concluir que se ha cumplido con los objetivos planteados, ya que se han analizado y estudiado las redes definidas por software partiendo de un aspecto teórico con aplicaciones ya existentes y generando variantes que están un paso adelante en lo que debe cumplir y demandan las SDN. Este trabajo no solo se desarrolló en un entorno de emulación sino que se implementa un entorno físico, en el cual se plasman las prácticas maś didácticas para comprender y evaluar estudios de SDN.

ANEXO A

Guías de Práctica:

A.1. Práctica 1: Introducción a escenarios de simulación para SDN

A.1.1. Objetivos:

- Instalar el emulador Mininet y los controladores POX, Ryu y ODL para SDN.
- Ejecutar topologías en Mininet.
- Configurar y comprobar el enlace del plano de datos con el plano de control.

A.1.2. Introducción:

En esta práctica se busca que el estudiante se familiarice con el simulador Mininet y los controladores POX, Ryu y ODL. Para esto se ejecutará una topología predeterminada en Mininet y se verificará que exista la conexión con los controladores SDN POX y Ryu. Para el caso del controlador SDN ODL se creará un *script* de Python que corresponde a una topología simple la que será ejecutada en Mininet y se verificará la conexión con el controlador evidenciando los resultados en la interfaz gráfica web. Mininet es un simulador para redes SDN que permite ejecutar topologías predeterminadas así como topologías personalizadas basadas en lenguaje Python. Facilita además la creación de un entorno de pruebas enlazable con controladores como POX, Ryu, Opendaylight y otros. En el caso de pretender la simulación de funcionalidades más avanzadas, como servidores de contenido, es necesario la implementación de máquinas virtuales. Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[1].
- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[2]
- Conexión a Internet para la instalación de librerías y repositorios
- Mininet.
- Controlador Ryu.
- Controlador POX.
- Controlador ODL.

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.1, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

A.1.3. Marco teórico a desarrollar:

- Emulador Mininet.
- Controlador Ryu.
- Controlador POX.
- Controlador ODL.

[1] Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop
[2] Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux

Tabla A.1: Software utilizado para el desarrollo de la guía Práctica 1

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 20.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.11.0
Virtualizador de *Switch*	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Controlador SDN	POX	0.7.0
Controlador SDN	ODL	Beryllium SR1
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6

A.1.4. Instrucciones:

1. **Instalar el controlador Ryu**[3]**:** Se debe considerar la ejecución bajo Python 3 mediante `pip3`
2. **Instalar Open vSwitch**[4]**:** Se recomienda la instalación desde los paquetes de la distribución de sistema operativo que se use, Debian para el caso de Ubuntu.
3. **Instalar Mininet**[5]**:** Se recomienda la instalación nativa desde la fuente a pesar de que el sitio oficial opta sugerir la instalación en máquina virtual. Debe considerar la nota importante respecto la instalación con Python 3[6]
4. **Conectar el controlador Ryu:**
   a) Desde una primera terminal crear una topología predefinida en Mininet, se recomienda comprender la sintaxis de la línea de comando 1 del Listado A.1 y ejecutarla.
   b) En una segunda terminal (o en la terminal `xterm c0` que abre el paso anterior) ejecutar el controlar Ryu con la aplicación de ejemplo `example_switch_13` mediante la línea de comando 2 del Listado A.1.
   c) En la terminal del controlador identificar los mensajes que evidencian la conexión con el *switch* `s1`, de igual manera en la terminal del *switch* `s1` visualizar la entrada de flujo creada hacia el controlador mediante la línea de comando 3 del Listado A.1.
   d) Desde la terminal del *host* 1, `h1`, probar la conectividad mediante ping al *host* 2, `h2`; luego viceversa, de `h2` a `h1`. Se recomienda las líneas de comando 4 y 5 del Listado A.1. La dirección IP de cada *host* se puede consultar por medio del comando `ifconfig` en la terminal respectiva. En el controlador se debe evidenciar los eventos `EventOFPPacketIn` en el primer ping que establece el enlace y se crean los flujos.
   e) Por último, revisar nuevamente las tablas de flujos en el *switch* `s1` mediante la línea de comando 3 del Listado A.1 y evidenciar las nuevas entradas de flujo.
   f) Detener el controlador (`Crtl+C` en la terminal del controlador) y cerrar Mininet mediante el comando `exit` en la terminal que ejecuta la topología. En algunas ocasiones en necesario limpiar el entorno de Mininet, se puede realizar mediante la línea de comando 6 del Listado A.1.

[3] Guía de instalación desde el repositorio oficial de Ryu: https://github.com/faucetsdn/ryu#whats-ryu
[4] Guía de instalación de Open vSwitch: https://docs.openvswitch.org/en/latest/intro/install/index.html#installation-from-packages
[5] Guía de Instalación nativa de Mininet: http://mininet.org/download/#option-2-native-installation-from-source
[6] Nota Importante de Mininet sobre Python 2 y Python 3: http://mininet.org/download/#important-note-python-2-and-python-3-mininet

```
user@t1:~$ sudo mn --topo minimal --mac --switch ovsk --controller
    remote -x #Crea topologia en Mininet
user@t2:~$ ryu-manager --verbose ryu.app.example_switch_13 #Ejecuta Ryu
     con app ejemplo
root@s1:/home/user# ovs-ofctl -O OpenFlow13 dump-flows s1 #Consulta de
    Tabla de Flujos en s1
root@h1:/home/user# ping -c10 10.0.0.2 #Ping de h1 a h2
root@h2:/home/user# ping -c10 10.0.0.1 #Ping de h2 a h1
user@t1:~$ sudo mn -c #Limpia el entorno de Mininet
```

Listado A.1: Líneas de Comando para conectar el controlador Ryu

5. **Conectar el controlador POX:** La instalación de Mininet realizada en numeral 3 de esta sección incluye por defecto a POX y se ubicada en el directorio `home`; en el caso de no existir, se puede instalar desde su repositorio oficial.[7]

   a) En una primera terminal ingresar al directorio `pox` y ejecutar en controlador POX con ciertos complementos básicos como muestra la línea de comando 1 del Listado A.2.

   b) Desde una segunda terminal crear una topología predefinida en Mininet mediante la línea de comando 1 del Listado A.2. De inmediato evidenciar en el controlador la conexión del *switch* `s1` (MAC).

   c) Realizar pruebas de conexión mediante ping, use la línea de comando 3 del Listado A.2. Mientras se ejecuta el ping se puede verificar las tablas de flujos temporales creadas en el *switch* `s1` mediante la línea de comando 4.

   d) Para crear entradas de flujo instaladas permanentemente en el *switch* `s1`, basadas en direcciones MAC, se debe cambiar el complemento que corre POX. Se detiene el controlador (`Crtl+C`) y se ejecuta la línea de comando 5 del Listado A.2.

   e) Ejecutar ping nuevamente, línea de comando 6

   f) Consultar nuevamente las tablas de flujo creadas en el *switch* `s1` mediante la línea de comando 4.

   g) Detener el controlador y cerrar Mininet.

```
user@t1:~/pox$ ./pox.py samples.pretty_log forwarding.l2_learning #
    Ejecuta POX con componentes para visualización y reenvió L2
user@t2:~$ sudo mn --topo minimal --mac --switch ovsk --controller
    remote -x #Crea topología en Mininet
root@h1:/home/user# ping 10.0.0.2 #Ping de h1 a h2
root@s1:/home/user# ovs-ofctl -O OpenFlow13 dump-flows s1 #Consulta de
    Tabla de Flujos en s1
user@t1:~/pox$ ./pox.py samples.pretty_log forwarding.l2_pairs #Ejecuta
     POX con componentes (visualización, reenvió L2 e instala entradas)
root@h1:/home/user# ping -c1 10.0.0.2 #Ping de h1 a h2, cantidad 1
```

Listado A.2: Líneas de Comando para conectar el controlador POX

6. **Descargar e instalar el controlador OpenDaylight versión Beryllium SR1:** La descarga se puede realizar desde el sitio oficial de OpenDaylight[8].

[7] Página Oficial de Documentación de POX https://noxrepo.github.io/pox-doc/html/

[8] Sitio oficial de descarga del controlador ODL Beryllium SR1: https://docs.opendaylight.org/en/stable-beryllium/getting-started-guide/installing_opendaylight.html

7. **Instalar JDK 8**: Para la instalación de JDK 8 abrir un terminal y ejecutar la línea de comando 1 del Listado A.3.
8. **Instalación del controlador ODL Beryllium SR1**:Para la instalación del controlador Open-Dayligth versión Beryllium se sigue la guía de instalación oficial proporcionada por ODL[9].
9. Instalar los módulos de configuración de la red de datos desarrollados por ODL. Los módulos de configuración a instalar son:
   - APIS de REST: Instala un conjunto de instrucciones que permiten que el controlador y las aplicaciones de red se comuniquen.
   - Plugin de OpenFlow: Asegura el soporte de protocolo OpenFlow en las versiones más recientes e indica al controlador el puerto TCP/UDP de comunicación con el dispositivo de red.
   - L2 switch: permite la comunicación con cualquier tipo de conmutador habilitado para OpenFlow.
   - Herramientas de Yang: es un conjunto de herramientas que soporta aplicaciones en Java en el controlador ODL.
   - SAL: Es el servicio de abstracción de capa usado para extraer información de red, lo que permite que el controlador adquiera estadísticas de estado de red y características de la topología.
   - D-Lux: Proporciona la interfaz gráfica web en donde se encuentra la configuración del controlador.

   Los comandos para instalar los módulos indicados se dan en las líneas $2-9$ del Listado A.3.
10. Desde una segunda terminal ejecutar la topología creada en un *script* de Python, mediante la línea de comando 10 del Listado A.3. Esta topología consta de un *switch* conectado a 3 *hosts*.
11. **Acceder a la interfaz gráfica del controlador ODL:** Para esto se accede a la dirección `http://localhost:8181/index.html`.
12. Realizar pruebas de conexión mediante el comando pingall, desde el emulador Mininet.
13. Visualizar las entradas de flujos generadas en `s1`, para esto se ejecuta el comando de la línea 11 de A.3.
14. Visualizar desde la interfaz gráfica del controlador la topología ejecutada y detalles del nodo `s1`.
15. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá realizar pruebas de conectividad entre los *hosts* de la red y visualizar las entradas de flujo generadas.

```
1  user@t1:~$  sudo apt install openjdk-8-jdk
2  opendaylight-user@root> feature:install odl-restconf-all
3  opendaylight-user@root> feature:install odl-openflowplugin-all
4  opendaylight-user@root> feature:install odl-l2switch-all
5  opendaylight-user@root> feature:install odl-mdsal-all
6  opendaylight-user@root> feature:install odl-dlux-all
7  opendaylight-user@root> feature:install odl-netconf-topology
8  opendaylight-user@root> feature:install odl-dlux-yangui
9  opendaylight-user@root> feature:install odl-l2switch-switch-ui
10 user@t2:~$ sudo python3 topologia_P1.py
```

[9]Guía de instalación de ODL Beryllium: https://docs.opendaylight.org/en/stable-beryllium/getting-started-guide/installing_opendaylight.html

```
root@s1:/home/user# ovs-ofctl -O OpenFlow13 dump-flows s1
```

Listado A.3: Líneas de Comando para instalar y conectar ODL Beryllium SR1

A.1.5. Verificación del funcionamiento y resultado esperados.

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ryu:
  - Ejecutar la topología predefinida en Mininet.
  - Ejecutar el controlador Ryu.
  - Revisar las entradas de flujo generadas en `s1`
  - Realizar pruebas de conectividad mediante pings.
  - Revisar las entradas de flujo generadas en `s1`
- POX:
  - Ejecutar el controlador POX.
  - Ejecutar la topología predefinida en Mininet.
  - Realizar pruebas de conectividad mediante pings.
  - Consultar las tablas de flujo generadas en `s1`.
- ODL:
  - Ejecutar el controlador ODL.
  - Ejecutar la topología creada.
  - Realizar pruebas de conectividad.
  - Visualizar la topología ejecutada en la interfaz gráfica del controlador.

1. Controlador Ryu
   - En una primera terminal se debe ejecutar la topología `minimal` de Mininet según el comando indicado en A.1. El resultado esperado se evidencia en la Figura A.1

Figura A.1: Salida de ejecutar la topología *minimal* de Mininet.

- En una segunda terminal se ejecuta la aplicación `example_switch_13` con el controlador Ryu con el comando indicado en A.1. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al controlador (`packet in`) o errores que pueda presentar la aplicación. En la Figura A.2 se puede ver una ejecución limpia, sin errores, y los registros de los primeros paquetes entrantes desde `s1` y `s2`. En el caso de recibir errores, se deben resolver los mismos antes de continuar.

```
jofnar@jofnar-HP:~$ ryu-manager --verbose ryu.app.example_switch_13
loading app ryu.app.example_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.example_switch_13 of ExampleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK ExampleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'ExampleSwitch13': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'ExampleSwitch13': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f364d993a90> address:('127.0.0.1', 43148)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f364d9abc70>
move onto config mode
EVENT ofp_event->ExampleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xd4a99ace,OFPSwitchFeatures(auxiliary_id=0,c
apabilities=79,datapath_id=1,n_buffers=0,n_tables=254)
move onto main mode
EVENT ofp_event->ExampleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
EVENT ofp_event->ExampleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
```

Figura A.2: Ejecución de la aplicación *example_switch_13* con el controlador Ryu y registros

- En este paso se ejecuta el comando de la línea 3 A.1 que permite obtener las entradas de flujo de `s1`. En la Figura A.3 se visualiza una entrada de flujo, donde la acción de salida indica que el paquete entrante se redirige hacia el controlador; esto demuestra que existe conexión entre *switch* y controlador permitiendo el intercambio de mensajes.

```
root@jofnar-HP:/home/jofnar# ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=265.558s, table=0, n_packets=2, n_bytes=140, priority=0 actions=CONTROLLER:65535
root@jofnar-HP:/home/jofnar#
```

Figura A.3: Entradas de flujo iniciales en *s1*

- A continuación se realizan pruebas de conectividad entre `h1` y `h2`, mediante ping, como se observa en la Figura A.4, existe conexión entre los dispositivos lo que conlleva a la generación de nuevas entradas de flujo que permitan dicha conexión.

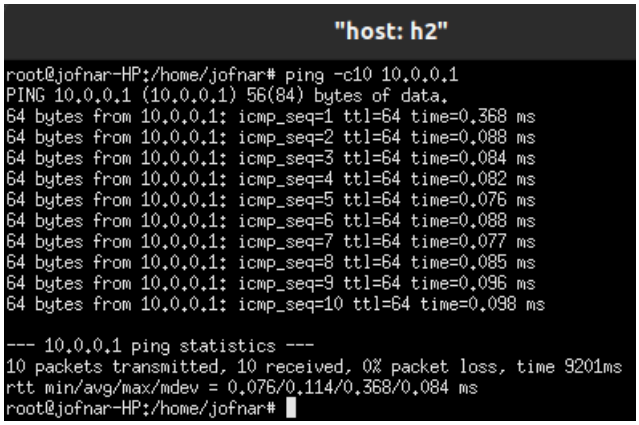


Figura A.4: Prueba de conectividad mediante ping

- Se vuelven a verificar la entradas de flujo de `s1`, el resultado esperado se visualiza en la Figura A.5. Como se observa se ha generado dos entradas de flujo que permiten la comunicación bidireccional entre los *hosts*. Estas se generan ya que al llegar un paquete al *switch*, si este no sabe qué hacer con el mismo, envía un mensaje `packet-in` al controlador y este responde con `packet-out` en donde se hallan las instrucciones para tratamiento del paquete, corroborando la conexión entre los mismos.

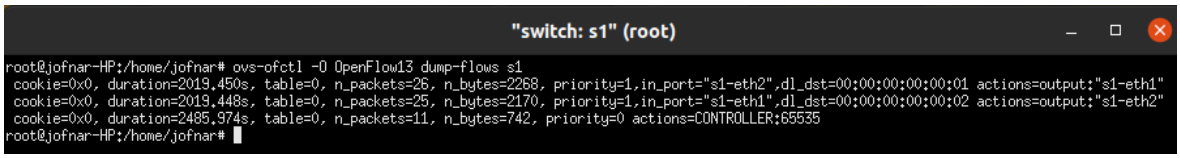


Figura A.5: Entradas de flujo en *s1*

2. Controlador POX

- En una primera terminal se debe ejecutar la topología `minimal` de Mininet según el comando indicado en A.2.
- En una segunda terminal se ejecuta el controlador POX con complementos básicos, para esto se utiliza la línea 1 del Listado A.2. A continuación se puede visualizar en los registros del controlador la conexión establecida entre `s1` y el controlador. Esto se evidencia en la Figura A.6

Figura A.6: Conexión entre *s1* y el controlador SDN POX.

- En este paso se realizan pruebas de conectividad mediante pings entre `h1` y `h2`, para lo que se ejecuta el comando de la línea 3 indicada en el Listado A.2 desde el terminal de `h1`; mientras se realizan las pruebas de conectividad se visualizan las entradas de flujo instaladas en `s1` que corresponden al tráfico que cursa entre los *hosts* en mención. Esto se evidencia en la Figura A.7

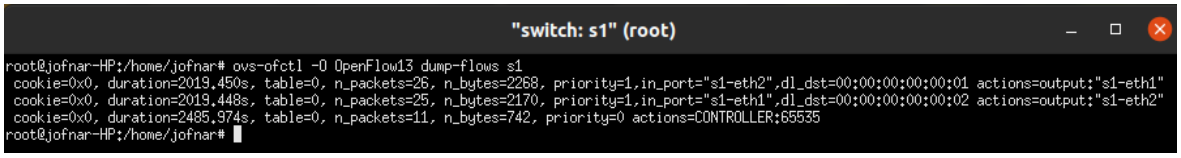


Figura A.7: Pruebas de conexión y entradas de flujo en *s1*

3. Controlador ODL

- **Ejecutar el controlador ODL:** Para ejecutar el controlador ODL, en una terminal se debe ubicar en el directorio en donde se encuentra la carpeta del controlador y desde allí se debe ejecutar la línea `sudo ./bin/karaf`, la ejecución del controlador se evidencia en la Figura A.8, ya en el controlador se realiza la instalación de los paquetes indicados en A.3.
- En una segunda terminal se debe ubicar en el directorio en donde se encuentre el *script* de Python, y ejecutar la topología con el comando indicado en la línea 10 del Listado A.3. Otra forma de verificar la correcta ejecución de la topología es mediante el comando `net`, que permite verificar la creación de los enlaces y elementos de la topología. La salida del comando `net` se evidencia en la Figura A.9
- A continuación se verifica la conectividad entre los elementos de la red, para lo cual se ejecuta el comando `pingall` en Mininet, el resultado obtenido se indica en la Figura A.10 con un `0%` de paquetes perdidos.

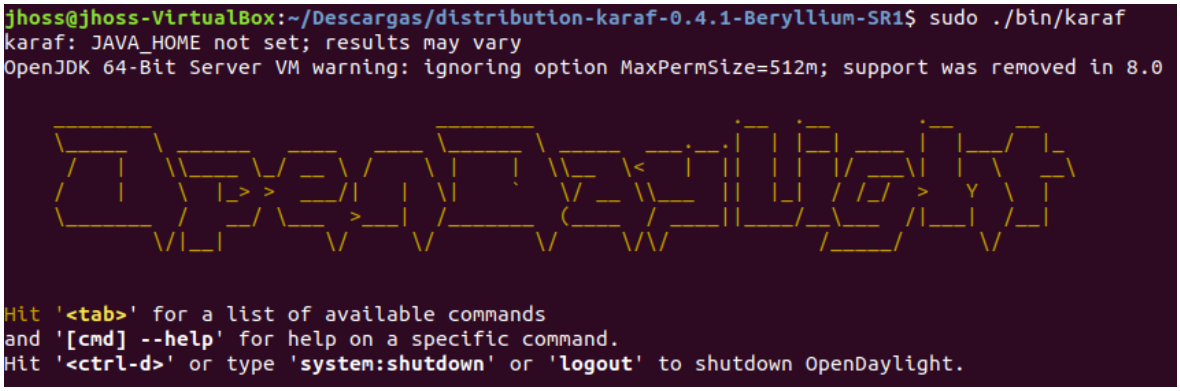


Figura A.8: Ejecución de controlador ODL

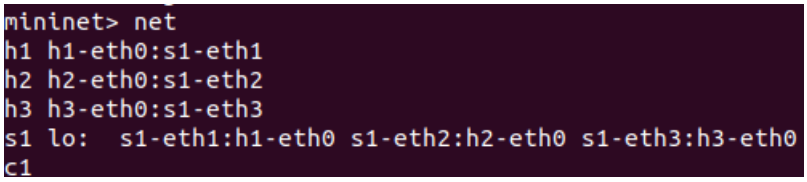


Figura A.9: Salida del comando *net* en Mininet

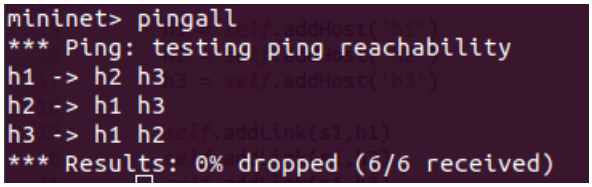


Figura A.10: Salida del comando *pingall* en Mininet

- Para corroborar la conexión entre el *switch* y controlador se verifican las entradas de flujo generadas en `s1`, según lo indicado en las instrucciones. El resultado se evidencia en la Figura A.11, en donde se observan 5 entradas de flujo que permiten la comunicación entre elementos de la red.

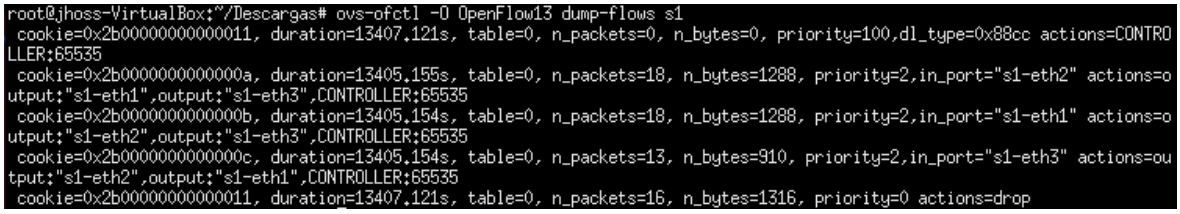


Figura A.11: Entradas de flujo en *s1*

- Desde la interfaz gráfica web del controlador ODL se puede observar la topología creada, y las estadísticas del nodo `s1` en donde se muestran los paquetes transmitidos, paquetes recibidos, bytes transmitidos y bytes recibidos en cada puerto del nodo `s1`. Esto se evidencia en la Figura A.12

Node Connector Statistics for Node Id - openflow:1

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:1:2	26	2720	1916	233128	0	0	0	0	0	0	0	0
openflow:1:1	26	2720	1916	233128	0	0	0	0	0	0	0	0
openflow:1:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0
openflow:1:3	26	2722	1916	233212	0	0	0	0	0	0	0	0

Figura A.12: Estadísticas en el nodo *s1*

A.1.6. Preguntas y resoluciones:

Esta práctica buscar responder las siguientes interrogantes:

- A nivel de complejidad al instalar y tras probar funcionalidades básicas: ¿Cuál controlador se presenta como el más complejo y cuál como el más sencillo?
- Considerando el lenguaje de programación en el que se basa cada controlador: ¿Que controlador sería más familiar y comprensible a la hora de programar?
- ¿Que controlador considera que presta mayores funcionalidades?

A.1.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción**, que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico**, de acuerdo a lo indicado previamente en A.1.3.
- **Desarrollo de la práctica:** Enumere los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos**, enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** En base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.2. Práctica 2: Topologías básicas y personalizadas en Mininet

A.2.1. Objetivos:

- Desarrollar destrezas en la elaboración de topologías tanto predeterminadas como personalizadas en el emulador Mininet.
- Ejemplificar la topología *Tree* de Mininet y visualizar el resultado de ejecutar comandos principales.
- Crear una topología personalizada usando el emulador Mininet.

A.2.2. Introducción:

En esta práctica se busca que el estudiante se familiarice con el simulador Mininet, para lo cual se ejemplifica la topología predeterminada de Mininet (*Tree*) y se visualiza el funcionamiento de comandos principales. También se creará una topología personalizada mediante un *script* de Python, la cual será conectada al controlador SDN y se visualizará la topología creada desde la interfaz de usuario del controlador ODL.

Mininet permite la creación de entornos SDN, la ventaja de este software es que permite llevar los entornos simulados a entornos físicos sin mucha complicación. Mininet admite topologías personalizadas y contiene una API basada en Python. Al ser un entorno extenso para la emulación de comportamientos de red existen comandos que permiten ejecutar diversas funcionalidades, algunos de los comandos más usados son:

- pingall: realiza pings entre todos los *hosts* de la topología.
- net: lista los nodos, enlaces e interfaces Ethernet conectados.
- dump: proporciona información sobre los elementos de red.
- xterm h1: abre una terminal xterm de h1.

El controlador SDN ODL cuenta con una interfaz de usuario DLUX situada en el plano de aplicaciones de red, la cual proporciona una gestión simple e intuitiva de la topología de red. Para acceder a esta interfaz se debe conectar a través de un navegador a la url `http://<dir_ip>:8181/index.html` en donde *dir_ip* es la dirección IP en donde se encuentra el controlador `ODL`. DLUX contiene el módulo `topology` que muestra una interpretación gráfica de la topología de red, también se puede instalar el módulo de `nodes` que proporciona estadísticas de red y puertos de los conmutadores.

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[1].
- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[2]
- Mininet.

[1]Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop
[2]Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux

- Controlador Ryu.
- Controlador ODL.

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.2. Sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.2: Software utilizado para el desarrollo de la guía Práctica 2

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.11.0
Virtualizador de Switch	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Controlador SDN	ODL	Beryllium SR1
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6

A.2.3. Marco teórico a desarrollar:

- Emulador Mininet.
- Comandos principales de Mininet.
- Topologías predeterminadas de Mininet.
- Topologías personalizadas en Mininet.

A.2.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

1. Ejecutar la topología *tree* de Mininet de tal manera que ésta se conecte a un controlador SDN, y las direcciones MAC se asignen de forma aleatoria pero ordenada.
2. Ejecutar los comandos principales de Mininet y analizar el resultado obtenido, los comandos a ejecutar son:
   - pingall
   - nodes
   - net
   - dump
   - xterm `h1`
   - link `s1 h1` down
   - link `s1 h1` up
   - iperf `h1 h2`
3. **Crear la topología personalizada**:
   - Crear el *script* en pyhton.
   - Importar librerías de Mininet para Python:
     - mininet.topo (Topo)
     - mininet.net (Mininet)

- mininet.log (setLogLevel)
- mininet.cli (CLI)
- mininet.node (RemoteController)

- **Añadir los *switches*:** Para añadir los *switches* se deben escribir líneas similares a: `s1=self.addSwitch('s1', protocols='OpenFlow13')`, en donde *'s1'* es el nombre que se le da al *switch*.
- **Añadir *hosts*:** Para añadir los *hosts* se deben escribir líneas similares a: `h1=self.addHost('h1',mac= ''00:00:00:00:00:01'', ip=''10.70.1.2/24'')`, en donde *'h1'* es el nombre del *host*; también se indica la dirección MAC e IP del *host*.
- Añadir links entre los elementos de la red de acuerdo a lo que se indica en la Figura A.13, para añadir los enlaces se escriben líneas similares a: `self.addLink(s1,s2,2,1)`, dicha línea crea el enlace entre `s1` (puerto 2) y `s2` (puerto 1).
- Desde la función main, establecer un controlador remoto, se debe especificar la dirección IP del controlador e inicializar la topología.

4. Ejecutar la topología personalizada elaborada.
5. Verificar su conexión con el controlador ODL y visualizar la topología en la interfaz gráfica web del controlador.

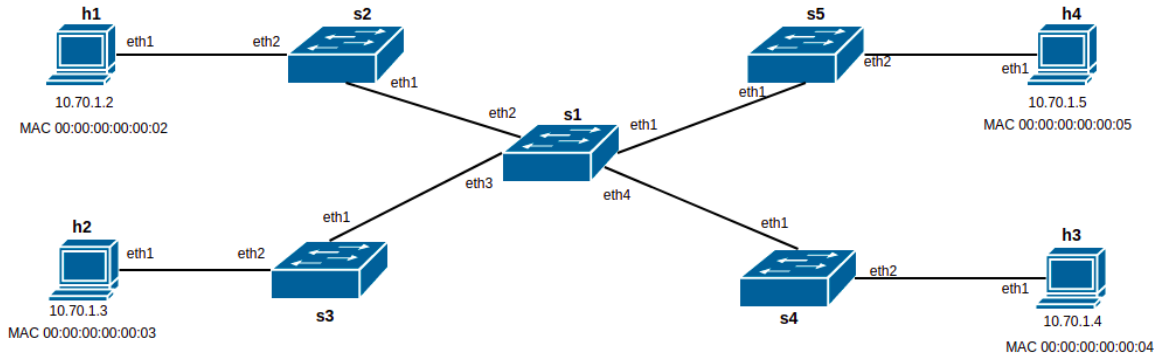


Figura A.13: Topología personalizada a implementar.

6. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá revisar si la topología ha sido creada de manera correcta mediante la ejecución de comandos principales en la CLI de Mininet y visualización en la interfaz gráfica web del controlador ODL.

A.2.5. Verificación de funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología *Tree* de Mininet.
- Ejecutar los comandos principales y analizar el resultado obtenido.
- Ejecutar la topología personalizada.
- Visualizar la topología desarrollada en la interfaz de usuario del controlador ODL.

1. En una primera terminal se debe ejecutar el controlador Ryu, como se realizó en la Práctica 1 lo que permite la conectividad entre el controlador y la topología a ejecutar, permitiendo realizar las pruebas respectivas de manera efectiva. En una segunda terminar se debe ejecutar el comando: `sudo mn --topo tree,2,2 --mac --switch ovsk --controller remote` en donde:
   - –topo: permite editar el tamaño y el tipo de topología que se va a emular, `tree,2,2` produce una topología en árbol compuesta de 2 niveles, 2 ramas y 2 *hosts* conectado a cada *switch* en las hojas.
   - –mac: permite asignar direcciones MAC de forma ordenada.
   - –switch: permite invocar un tipo de *switch*, en este caso se utiliza `ovsk` que usa Open vSwitch en modo kernel para cada *switch*.
   - –controller: permite llamar un controlador, `remote` permite el uso de un controlador compatible con OpenFlow y externo a Mininet.

   El resultado obtenido se evidencia en la Figura A.14, en donde se demuestra la conexión entre controlador y topología, iniciando el intercambio de mensajes.

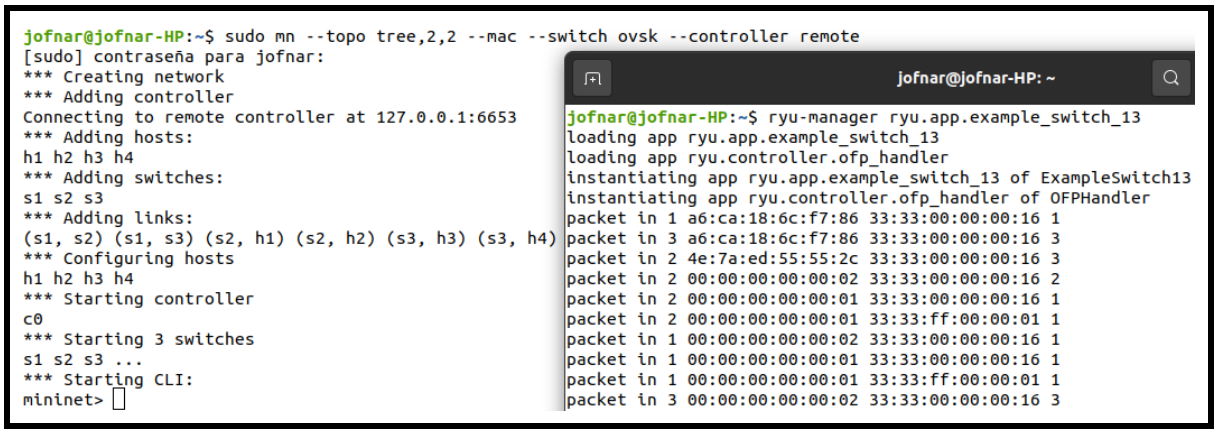


Figura A.14: Resultado de ejecutar la topología *Tree* y conexión con el controlador Ryu.

2. Desde el terminal de Mininet, se debe ejecutar los comandos indicados en la sección de instrucciones, algunos de los resultados se evidencian en las Figuras A.15, A.16, A.17 y A.18
   - pingall: realiza pings entre los *hosts* de la topología.
   - nodes: lista los elementos de red, (*switches*).
   - net: Lista nodos, enlaces e interfaces conectadas.
   - dump: indica información acerca de los elementos de red.
   - link `s1 h1` down: deshabilita el enlace entre `h1` y `s1`.
   - link `s1 h1` up: habilita el enlace entre `h1` y `s1`.
   - iperf `h1` y `h2`: mide el máximo ancho de banda entre dos *hosts* basado en TCP.
   - xterm `h1`: abre una terminal independiente de `h1`.

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
c0
```

Figura A.15: Resultado de ejecutar el comando net en el terminal de Mininet.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=20189>
<Host h2: h2-eth0:10.0.0.2 pid=20191>
<Host h3: h3-eth0:10.0.0.3 pid=20193>
<Host h4: h4-eth0:10.0.0.4 pid=20195>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=20200>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=20203>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=20206>
<RemoteController c0: 127.0.0.1:6653 pid=20183>
```

Figura A.16: Resultado de ejecutar el comando *dump* en el terminal de Mininet.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['32.1 Gbits/sec', '32.1 Gbits/sec']
```

Figura A.17: Resultado de ejecutar el comando *iperf* en el terminal de Mininet.

Figura A.18: Resultado de ejecutar el comando *xterm* en el terminal de Mininet.

En este punto el estudiante debe ejecutar todo los comandos indicados y ahondar en el análisis de los resultados obtenidos.

3. **Ejecutar la topología personalizada:** En una terminal se debe ejecutar el controlador ODL según lo indicado en la Práctica 1 y en una segunda terminal ubicarse en la dirección en donde se encuentre el *script* de Python creado y ejecutar la topología con el comando: `sudo python3 topologia_P2.py`. En la Figura A.19 se indica el resultado de la ejecución del comando `net` que permite visualizar la lista de nodos, enlaces e interfaces conectadas.

```
mininet> net
h1 h1-eth1:s2-eth2
h2 h2-eth1:s3-eth2
h3 h3-eth1:s4-eth2
h4 h4-eth1:s5-eth2
s1 lo:  s1-eth1:s5-eth1 s1-eth2:s2-eth1 s1-eth3:s3-eth1 s1-eth4:s4-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:h1-eth1
s3 lo:  s3-eth1:s1-eth3 s3-eth2:h2-eth1
s4 lo:  s4-eth1:s1-eth4 s4-eth2:h3-eth1
s5 lo:  s5-eth1:s1-eth1 s5-eth2:h4-eth1
c0
```

Figura A.19: Resultado de ejecutar el comando *net* en el terminal de Mininet.

4. En este paso se accede a la interfaz gráfica web del controlador `ODL`, en la sección `Topology` se observa la topología implementada que surge de algoritmos internos que ejecuta ODL para el descubrimiento de rutas. Esto se evidencia en la Figura A.20.

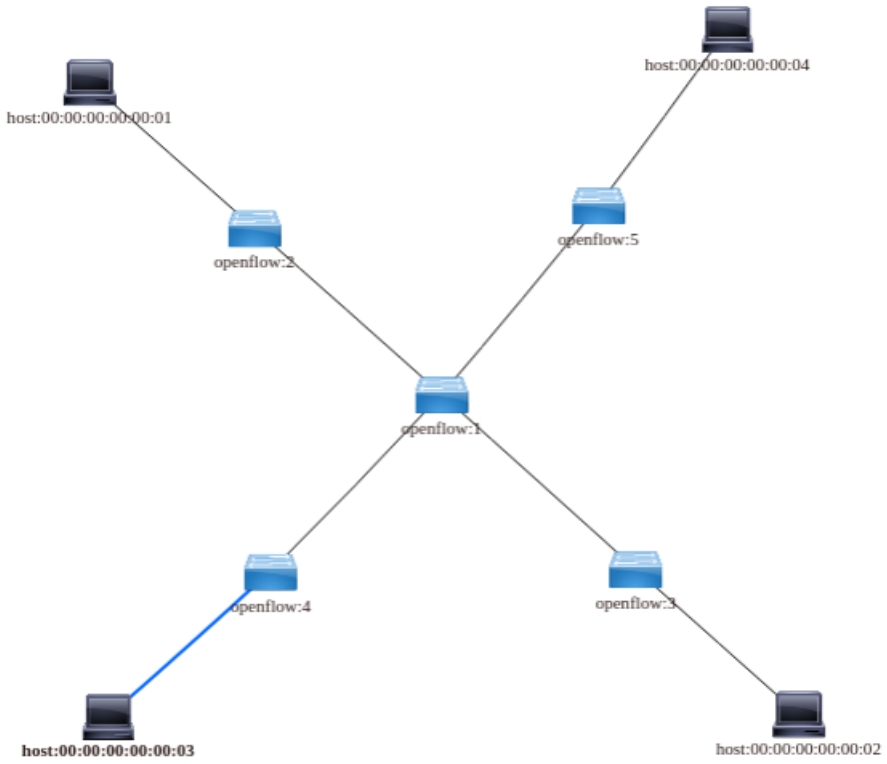


Figura A.20: Visualización de la topología implementada desde la interfaz del controlador ODL.

Dentro de la misma interfaz gráfica del controlador en la sección de nodos se puede visualizar los nodos de la red y sus estadísticas, como se visualiza en la Figura A.21.

Node Id	Node Name	Node Connectors	Statistics
openflow:2	s2	3	Flows | Node Connectors
openflow:3	s3	3	Flows | Node Connectors
openflow:4	s4	3	Flows | Node Connectors
openflow:5	s5	3	Flows | Node Connectors
openflow:1	s1	5	Flows | Node Connectors

Figura A.21: Lista de Nodos visualizados desde el controlador ODL.

A.2.6. Preguntas y resoluciones:

Esta práctica busca responder las siguientes interrogantes:

- ¿Qué limitaciones tienen las topologías predeterminadas del emulador Mininet?
- ¿Qué tipo de información se puede consultar mediante los comandos de Mininet?
- ¿Qué funcionalidad permite ejecutar la CLI de Mininet?

A.2.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.2.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** En base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.3. Práctica 3: Implementación de agregación de enlaces con Ryu

A.3.1. Objetivos:

- Diseñar una topología que permita configurar la implementación de agregación de enlaces.
- Configurar dos interfaces de grupo de enlace sobre un *switch* conectado a otros *switches* y *hosts*
- Habilitar LACP mediante Ryu y configurar interfaces `bond` sobre *hosts* que actúan como servidores.
- Evidenciar la distribución de carga de tráfico que genera la agregación de enlace.
- Comprobar la tolerancia a fallos de la red ante la caída de uno de los enlaces del grupo de enlace.

A.3.2. Introducción:

En esta práctica se busca implementar la agregación de enlaces dentro de una topología personalizada en Mininet que luego pueda implementarse en un entorno de pruebas físicas. Se pretende evaluar ciertas características que implica la agregación de enlace como es QoS por mayor ancho de banda, tolerancia a fallos y distribución de carga de tráfico.

La tecnología de agregación de enlaces está definida en la IEEE802.1AX-2008, la cual permite combinar múltiples interfaces físicas en un solo enlace lógico. Esta implementación permite incrementar la velocidad (ancho de banda) entre dispositivos de la red al mismo tiempo asegura redundancia y por tanto tolerancia a fallos.

Previamente en los dispositivos de red involucrados se debe configurar las interfaces que pertenecerán al grupo de enlace. Existen 2 métodos para implementar la agregación de enlaces:

- Método estático: Cada elemento de la red es instruido directamente.
- Método dinámico: Usa el protocolo de control de agregación de enlaces (LACP)

En esta práctica se plantea el uso del método dinámico, el mismo que provoca que las interfaces del grupo de enlace que están en contra parte intercambien periódicamente unidades de datos LACP para verificar que la comunicación aún está disponible. En el caso de que se interrumpa este intercambio, significaría que ocurrió un fallo y el dispositivo deja de estar disponible por medio de esa interfaz por lo que el envío y recepción de paquetes se destina solo por las interfaces restantes.

La topología planteada para esta práctica se visualiza en la Figura A.22, al finalizar esta práctica el estudiante habrá configurado grupos de enlace sobre un *switch* mediante el controlador Ryu, así como habilitará interfaces `bond` sobre *hosts* adyacentes que actúa como servidores.

La práctica permitirá verificar la distribución de carga del tráfico proveniente de `h2`, `h3` y `h4` hacia `h1` y viceversa, es decir, se visualizará que el tráfico desde y hacia `h1` fluye por las interfaces del Grupo de Enlace de `s1` (algunos casos por `eth1` y otros por `eth2`).

Por otra parte se puede dar de baja una de las interfaces del grupo de enlace, por ejemplo, `h1-eth0` contraparte de `s1-eth1`. Esto, con el fin de verificar que, pasados ciertos segundos, el controlador detecta es estado inactivo del enlace y sin embargo la comunicación hacia `h1` se mantiene debido a la interfaz restante aún conectada y activa.

Para el desarrollo de esta práctica se requiere disponer de:

- Ordenador con alguna distribución de Sistema Operativo basada en Linux, se recomienda Ubuntu en versiones LTS[1].
- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[2]
- Simulador Mininet.
- Controlador Ryu.

Esta guía se ha desarrollado considerando el software más actualizado, estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.3, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.3: Software utilizado para el desarrollo de la guía Práctica 3

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 20.04, kernel 5.11.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de Switch	Open vSwitch	2.13.3
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6

A.3.3. Marco teórico a desarrollar:

- Agregación de interfaces de red (*Ethernet Bonding*).
- Protocolo de control de agregación de enlace (LACP).
- Principales beneficios de la agregación de enlace.

A.3.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento y apoyarse simultáneamente en el Capítulo 5, *Link Aggregation*, del Libro RYU SDN Framework[3]:

1. **Crear la topología:** La topología recomendada a implementar en Mininet se muestra en la Figura A.22. Para esto se debe desarrollar un *script* en Python, `topologiaLA.py`. La topología consta de 3 *switches* y 4 *hosts*, dos de ellos presentan enlaces redundantes a un *switch*.
2. **Escribir la aplicación para el controlador Ryu:** Para el desarrollo de la aplicación en lenguaje Python, denominada `agregación.py`, se toma como referencia una aplicación de ejemplo incluido en Ryu para los fines de agregación de enlace. Esta aplicación se encuentra documentada y para adaptarla a la topología de la Figura A.22 se debe tomar en cuenta las siguientes instrucciones:
   a) Revisar la aplicación `simple_switch_lacp_13.py` presentada y explicada en RyuBook[3]. La aplicación muestra la estructura básica del código en Python que permite implementar la agregación de enlace.

[1]Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop
[2]Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux
[3]RyuBook: https://osrg.github.io/ryu-book/en/Ryubook.pdf

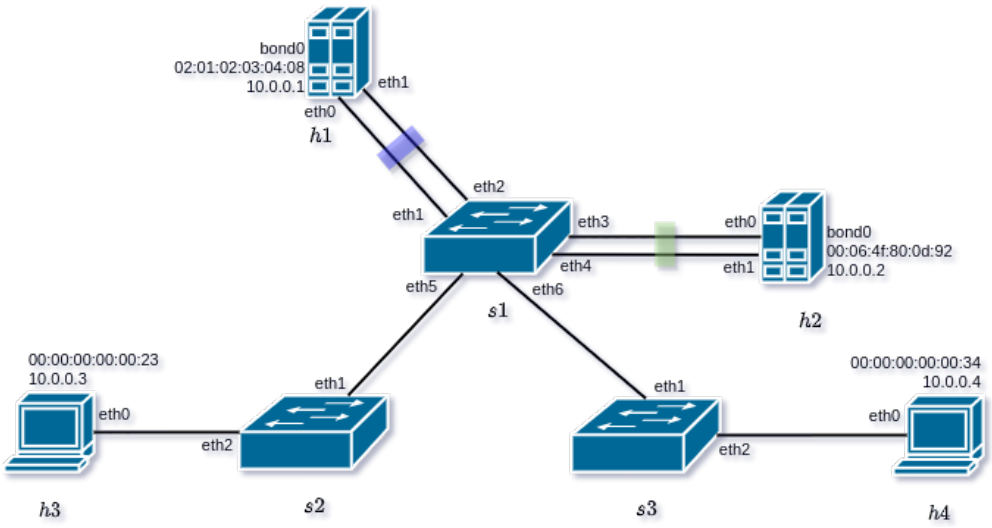


Figura A.22: Topología para Agregación de Enlace.

b) Clonar (copiar o transcribir) completamente el código de `simple_switch_lacp_13.py` en el *script* `agregación.py`.

c) Agregar el segundo grupo de enlace con las interfaces `eth3` y `eth4` para el *switch* `s1` en el nuevo *script*. Por defecto dentro de la función `__init__` se encuentra creado el primer grupo de enlace con las interfaces `eth1` y `eth2` en el mismo *switch*, evidenciado en la línea 1 del Listado A.4, entonces se debe incluir la línea 2 del mismo Listado A.4.

```
1 self._lacp.add(dpid=str_to_dpid('0000000000000001'),ports=[1,2])
2 self._lacp.add(dpid=str_to_dpid('0000000000000001'),ports=[3,4])
```

Listado A.4: Líneas de código para la Agregación de dos Enlaces de Grupo.

3. **Preparar configuraciones complementarias:** Se requieren líneas de comando para la configuración de la agregación de enlace en los *hosts* `h1` y `h2`, se recomienda crear un *script* bash para una rápida ejecución.

   - Para el caso de `h1` se presenta un ejemplo en el Listado A.5 según la documentación presentada por RyuBook[3], guardado como `bond_h1.sh` en un directorio de su preferencia.
   - Para `h2` se recomienda crear el *script* `bond_h2.sh`, donde se debe modificar para incluir las interfaces y direcciones MAC correspondientes a `h2`. Además debe tener especial consideración en la dirección de la interfaz `bond0`, se recomienda la MAC 00:06:4f:80:0d:92.

```
1  echo "alias bond0 bonding" > /etc/modprobe.d/bonding.conf
2  echo "options bonding mode=4" >> /etc/modprobe.d/bonding.conf
3  modprobe bonding
4  ip link add bond0 type bond
5  ip link set bond0 address 02:01:02:03:04:08
6  ip link set h1-eth0 down
7  ip link set h1-eth0 address 00:00:00:00:00:11
8  ip link set h1-eth0 master bond0
9  ip link set h1-eth1 down
10 ip link set h1-eth1 address 00:00:00:00:00:12
```

```
ip link set h1-eth1 master bond0
ip addr add 10.0.0.1/8 dev bond0
ip addr del 10.0.0.1/8 dev h1-eth0
ip link set bond0 up
```

Listado A.5: Script bond_h1.sh para la configuración de la interfaz bond0 en h1

- Otorgar permisos de ejecución al usuario en los dos *scripts* creados previamente, por ejemplo: `chmod u+x bond_h1.sh`

4. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá realizar las configuraciones necesarias de la agregación de enlace y pruebas de conectividad mediante `ping` bajo distintas consideraciones.

A.3.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Configurar la agregación de enlace sobre los *hosts* `h1` y `h2`.
- Ejecutar el controlador con la aplicación desarrollada.
- Verificar la conexión entre *hosts*.
- Analizar la distribución de tráfico en el grupo de enlace entre `s1` y `h1`, luego en el Grupo de Enlace entre `s1` y `h2`.
- Comprobar la tolerancia a fallos del grupo de enlace entre `s1` y `h1`.

1. En una primera terminal se debe ubicar el directorio en el que se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `sudo python topologiaLA.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.23, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados. Los marcadores rojos de la Figura A.23 indican los enlaces redundantes hacia `h1` mientras que los marcadores azules indican aquellos hacia `h2`.

```
mininet> net
h1 h1-eth0:s1-eth1 h1-eth1:s1-eth2
h2 h2-eth0:s1-eth3 h2-eth1:s1-eth4
h3 h3-eth0:s2-eth2
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h1-eth1 s1-eth3:h2-eth0 s1-eth4:h2-eth1 s1-eth5:s2-eth1 s1-eth6:s3-eth1
s2 lo:  s2-eth1:s1-eth5 s2-eth2:h3-eth0
s3 lo:  s3-eth1:s1-eth6 s3-eth2:h4-eth0
c0
```

Figura A.23: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. A continuación se configuran los grupos de enlace en los *hosts* `h1` y `h2` para lo cual, mediante la CLI de Mininet, se abren terminales para cada *host*. En la terminal `xterm` de `h1` se ubica el directorio de almacenamiento del *script* de configuración y desde allí se ejecuta el mismo mediante `./bond_h1.sh`. De manera similar se procede en `h2`, desde el directorio correspondiente se ejecuta

`./bond_h2.sh`. La configuración correcta se evidencia al ejecutar el comando `ifconfig` en la terminal del *host*, la salida esperada para `h1` se muestra en la Figura A.24 donde se visualiza que las interfaces físicas `eth0` y `eth1` (*SLAVES*) comparten la misma dirección MAC que la interfaz lógica `bond0` (*MASTER*), que a su vez es la única interfaz con una IP asignada.

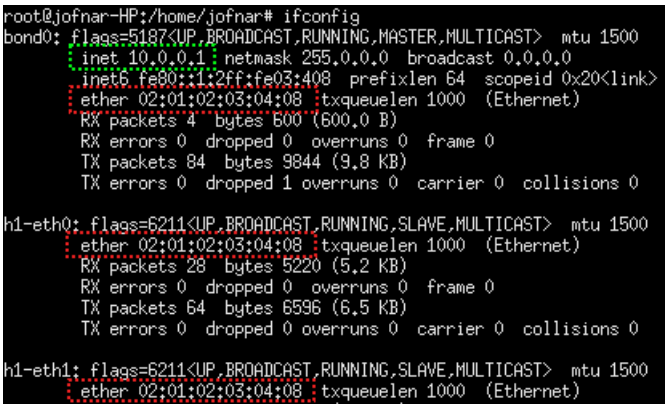


Figura A.24: Verificación de la Configuración para Agregación de Enlace en h1

De la misma forma, el estudiante deberá verificar la configuración para `h2`. De manera opcional se puede consultar el estado del controlador de enlace desde la terminal del *host* respectivo, mediante `cat /proc/net/bonding/bond0`.

3. Una vez configurados los grupos de enlace por el lado de los *hosts* y antes de ejecutar el controlador, se debe asegurar que los conmutadores empleen el protocolo OpenFlow versión 1.3. Con este fin, se abre las terminales `xterm` de cada uno de los *switches* y se ejecuta la línea de comando: `ovs-vsctl set Bridge s1 protocols=OpenFlow13`, donde `s1` corresponde al identificador del *switch*.
4. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `ryu-manager agregacion.py`. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al controlador (`packet in`) o errores que pueda presentar la aplicación. De manera particular, esta aplicación muestra registros de LACP sobre el intercambio de mensajes en las interfaces del grupo de enlace. En la Figura A.25 se puede

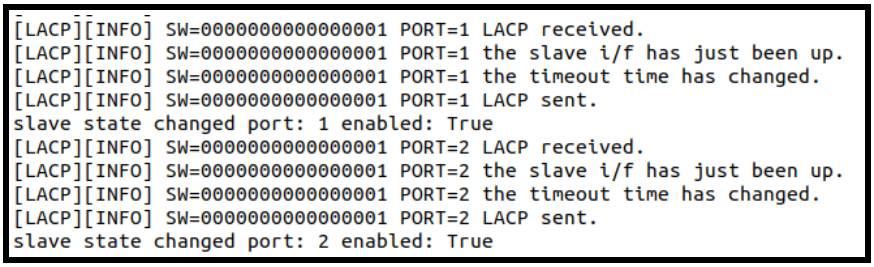


Figura A.25: Registros de Intercambio de Mensajes LACP tras la ejecución de la aplicación agregacion.py

ver parte de la ejecución de la aplicación `agregacion.py`, se muestran los registros de LACP correspondientes al primer grupo de enlace, conformadas por las interfaces `eth1` (`PORT=1`) y `eth2` (`PORT=2`) de `s1`. En el caso de recibir errores, se deben resolver los mismos antes de continuar. Los registros de LACP de la Figura A.25 se visualizan gracias a entradas de flujo instala-

das en `s1` con acciones que redirigen el tráfico de mensajes LACP (`0x809`) hacia el controlador. Esto se puede verificar al consultar las tablas de flujo en el *switch* con el comando `ovs-ofctl -O openflow13 dump-flows s1`, tal como se puede observar en la Figura A.26, las 2 entradas en recuadro rojo corresponden al primer grupo de enlance y las 2 remarcadas de azul corresponden al segundo. Adicionalmente se puede visualizar la alta prioridad de estas entradas (`65509`) y el tiempo que estas pueden permanecer activas sin encontrar una coincidencia (`idle_timeoout=90`).

```
root@jofnar-HP:/home/jofnar# ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=1565.600s, table=0, n_packets=51, n_bytes=6324, idle_timeout=90, send_flow_rem priority=65535,
in_port="s1-eth4",dl_src=00:00:00:00:00:14,dl_type=0x8809 actions=CONTROLLER:65509
 cookie=0x0, duration=1565.599s, table=0, n_packets=51, n_bytes=6324, idle_timeout=90, send_flow_rem priority=65535,
in_port="s1-eth3",dl_src=00:00:00:00:00:13,dl_type=0x8809 actions=CONTROLLER:65509
 cookie=0x0, duration=1559.142s, table=0, n_packets=59, n_bytes=7316, idle_timeout=90, send_flow_rem priority=65535,
in_port="s1-eth1",dl_src=00:00:00:00:00:11,dl_type=0x8809 actions=CONTROLLER:65509
 cookie=0x0, duration=1558.830s, table=0, n_packets=60, n_bytes=7440, idle_timeout=90, send_flow_rem priority=65535,
in_port="s1-eth2",dl_src=00:00:00:00:00:12,dl_type=0x8809 actions=CONTROLLER:65509
 cookie=0x0, duration=1587.655s, table=0, n_packets=30, n_bytes=2952, priority=0 actions=CONTROLLER:65535
```

Figura A.26: Entradas de flujo en s1 que envían los mensajes LACP hacia el controlador

5. El siguiente paso consiste en verificar la correcta conectividad entre todos los *hosts* de la red. Para esto, desde la CLI de Mininet se puede ejecutar `pingall`. El resultado esperado se muestra en la Figura A.27.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura A.27: Resultado exitoso de la salida del comando pingall

```
 cookie=0x0, duration=212.447s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",
dl_dst=02:01:02:03:04:08 actions=output:"s1-eth2"
 cookie=0x0, duration=212.444s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",
dl_dst=00:06:4f:80:0d:92 actions=output:"s1-eth3"
 cookie=0x0, duration=212.382s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth5",
dl_dst=02:01:02:03:04:08 actions=output:"s1-eth2"
 cookie=0x0, duration=212.380s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",
dl_dst=00:00:00:00:00:23 actions=output:"s1-eth5"
 cookie=0x0, duration=212.366s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth6",
dl_dst=02:01:02:03:04:08 actions=output:"s1-eth2"
 cookie=0x0, duration=212.364s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",
dl_dst=00:00:00:00:00:34 actions=output:"s1-eth6"
 cookie=0x0, duration=212.343s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth5",
dl_dst=00:06:4f:80:0d:92 actions=output:"s1-eth4"
 cookie=0x0, duration=212.341s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth4",
dl_dst=00:00:00:00:00:23 actions=output:"s1-eth5"
 cookie=0x0, duration=212.329s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth6",
dl_dst=00:06:4f:80:0d:92 actions=output:"s1-eth4"
 cookie=0x0, duration=212.327s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",
dl_dst=00:00:00:00:00:34 actions=output:"s1-eth6"
 cookie=0x0, duration=212.303s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth6",
dl_dst=00:00:00:00:00:23 actions=output:"s1-eth5"
 cookie=0x0, duration=212.298s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth5",
dl_dst=00:00:00:00:00:34 actions=output:"s1-eth6"
```

Figura A.28: Entradas de flujo en s1 creadas tras prueba de conectividad

La prueba de conectividad agrega nuevas entradas de flujo, se vuelve a consultar las tablas de flujo en `s1`, obteniendo el registro que se muestra en la Figura A.28, donde se han remarcado

por colores las interfaces de los grupos de enlace involucradas en las nuevas entradas de flujo. En este punto, el estudiante deberá ahondar en la explicación de las tablas de flujo esperadas y obtenidas para cumplir con los objetivos la presente práctica.

6. **Análisis de la distribución de tráfico**: En base a las entradas de flujo mostradas en la Figura A.28 se puede evaluar e interpretar la distribución de tráfico como se observa en la Figura A.29. Gráficamente se puede verificar que existe distribución de tráfico, en ambos grupos de enlace se emplean los enlaces paralelos abstraídos como uno solo. En cada grupo de enlace se puede notar que en algunos casos se usa uno de los enlaces para tráfico ascendente y el otro (su paralelo) para descendente; en otros casos el mismo enlace permite tráfico bidireccional.

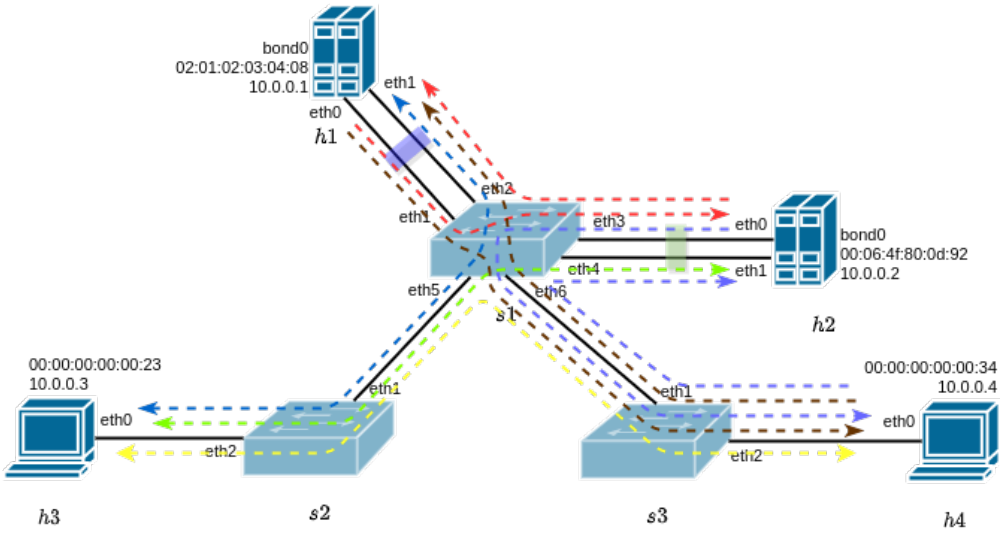


Figura A.29: Distribución de Tráfico a través de los Grupos de Enlace

Para cada grupo de enlace, el estudiante podría tabular lo obtenido en la Figura A.28 que corresponde lo ilustrado en la Figura A.29, considerando *host* origen, *host* destino, el puerto involucrado del *switch* y si el enlace es descendente, ascendente o bidireccional. Esto ayudaría a comprender aún más y contabilizar los enlaces que que usan uno u otro puerto del grupo de enlace.

7. **Análisis de la tolerancia a fallos**: Para evaluar esta funcionalidad es necesario separar una de las interfaces que componen el grupo de agregación de enlace. En la emulación, se puede realizar desde el *host*, en `h1` se dará de baja la interfaz `eth1` puesto que es la que maneja mayor cantidad de flujos de tráfico. Previamente se realiza `ping` de manera continua desde `h3` hacia `h1` y se inhabilita en `h1` su interfaz `eth1` mediante `ip link set h1-eth1 nomaster`.
   El comportamiento esperado se muestra en la Figura A.30, tras inhabilitar la interfaz, por parte del controlador se registra el evento de caducidad del tiempo de espera para el intercambio de mensajes LACP por lo que, asume que se perdió el enlace paralelo e inhabilita el puerto 2 del *switch* (contraparte de la interfaz separada `eth1`). Por otra parte, la conectividad por `ping` se detiene (`icmp_seq=22`) y luego se recupera (`icmp_seq=112`), habiendo pasado un tiempo bastante considerable.
   Para complementar la evaluación del comportamiento de respuesta de la tolerancia a fallos, se consulta nuevamente las tablas de flujo en `s1`, el resultado se muestra en la Figura A.31. Se puede

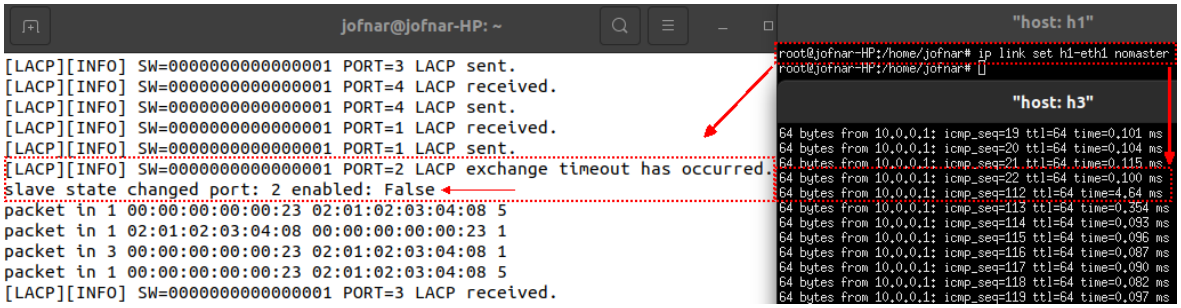


Figura A.30: Separación de la interfaz eth1 de h1 y comportamiento esperado

evidenciar que todas las entradas previas fueron borradas y se instalaron únicamente las entradas para el tráfico activo en ese momento (`h3-h1`, recuadro rojo). Nuevas entradas se instalarían a medida que se demande conectividad entre los *hosts*. El recuadro azul muestra las entradas de flujo hacia el controlador debido al intercambio de mensajes LACP, antes de la pérdida del enlace eran 4 (Figura A.26) y ahora corresponden 3. Todo esto, confirma el comportamiento de tolerancia a fallos que proporciona la agregación de enlace.

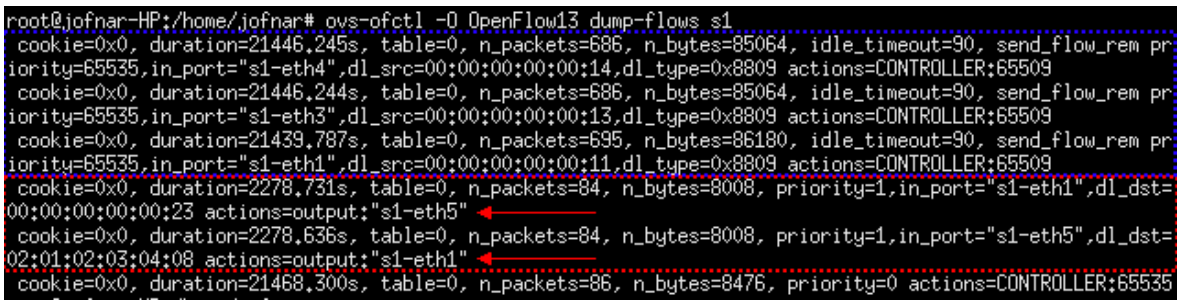


Figura A.31: Tabla de Flujos en s1 tras la respuesta por Tolerancia a Fallos

A.3.6. Preguntas y resoluciones:

Una vez realizada la práctica se puede responder las siguientes interrogantes:

- ¿Distribución de carga de tráfico es igual que balanceo de carga?
- ¿La tolerancia a fallos es igual que reenrutamiento rápido?
- Ante la pérdida de un enlace ¿A qué se debe la demora de la red para restablecer la conectividad por el enlace paralelo restante?
- En esta práctica se ha trabajado con agregación de enlace entre host-*switch*, ¿Se podría realizar agregación de enlace entre dos *switches*?

A.3.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.

- **Marco teórico:** de acuerdo a lo indicado previamente en A.3.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.4. Práctica 4: Ingeniería de tráfico con Ryu.

Manejo de tablas de grupo para división porcentual de tráfico

A.4.1. Objetivos:

- Entender el manejo de tabla de grupos.
- Dividir el tráfico para diferentes rutas.
- Añadir flujos según parámetros de coincidencia y acciones a realizar.
- Crear una topología personalizada.
- Comprender la división de tráfico y flujos añadidos.

A.4.2. Introducción:

En esta práctica se busca realizar una subdivisión porcentual de tráfico, es decir, se definirán 2 rutas, en donde por la ruta compuesta por `s1-s2-s3` transitará el 35 % del tráfico total y por la ruta compuesta por `s1-s3` circulará el 65 % del tráfico total. Para cumplir con lo establecido se hará uso de tablas de grupo. La división de tráfico en aplicaciones reales permitiría no colapsar una sola ruta (mejor camino) lo que ocasionaría lentitud en el servicio y hasta pérdida de paquetes. Esta práctica se basa en información impartida en el curso `SDN Crash Course (OpenFlow, Mininet, Ryu)` disponible en [1], acerca de tablas de grupo, en esta práctica se da un enfoque diferente considerando lo ya mencionado.

Un controlador SDN se comunica con el *switch* mediante el protocolo OpenFlow, a su vez los conmutadores pueden comunicarse con otros conmutadores y dispositivos finales. OpenFlow define tres tipos de tablas:

- Tabla de flujo: Hace que un paquete entrante coincida con un flujo e indica la acción a realizar con dicho paquete.
- Tablas de medidores: Incluye acciones relacionadas con el rendimiento de un flujo.
- Tablas de grupo: Permite representar un conjunto de puertos como una sola entidad para el reenvío de paquetes.

Una tabla de flujo puede dirigir un flujo hacia una tabla de grupo, provocando acciones que afecten a uno o más flujos. Al indicar una acción de grupo en un flujo se indica que el paquete va a ser procesado a través de un grupo en específico. La tabla se compone de *buckets*; cada *bucket* posee una lista de acciones que pueden ser aplicadas a los paquetes que ingresan. Existen diferentes tipos de grupo según la aplicación que se desee realizar, los tipos de grupo son: ALL, SELECT, INDIRECT y FAST-FAILOVER[2] El grupo ALL toma cualquier paquete entrante y los duplica hacia cada *bucket* para que estos lo operen de manera independiente. El grupo SELECT posee un peso asignado y cada paquete entrante ingresa a un solo depósito, la selección de *bucket* depende de la implementación del conmutador, es decir, no está definida. Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[3].

[1] Curso SDN Crash Course: `udemy.com`
[2] Libro Fundations Modern Networking: `http://williamstallings.com/Network/`
[3] Página oficial de descarga de Ubuntu: `https://ubuntu.com/download/desktop`

- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[4]
- Controlador RYU.
- Mininet.
- Herramienta iPerf.

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.4, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.4: Software utilizado para el desarrollo de la guía Práctica 4

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *switch*	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

A.4.3. Marco teórico a desarrollar:

- OpenFlow v1.3 mensajes y estructuras básicas.
- Tablas de flujo y su estructura.
- Tablas de grupo y su estructura.
- Herramienta iPerf: Funcionalidad y comandos (clientes en paralelo).

A.4.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

1. **Crear la topología:** La topología personalizada a implementar en Mininet se muestra en la Figura A.32, para lo cual se debe desarrollar un *script* en Python, `topologia_65_35.py`. La topología consta de 3 *switches*, 2 *hosts*, y dos rutas alternativas.
2. **Escribir la aplicación para el controlador Ryu:** Para el desarrollo de la aplicación en lenguaje Python, denominada `division_porcentual.py`, se debe tomar en cuenta las siguientes instrucciones:
   a) Revisar la aplicación `SimpleSwitch13` presentada y explicada en el Libro RYU SDN Framework[5]. La aplicación muestra la estructura básica del código en Python que permite desarrollar aplicaciones de mayor complejidad.
   b) Revisar la documentación de OpenFlow mensajes y estructuras [6], en donde se hallan ejemplificaciones de funciones tales como `switch_features_handler`, `send_group_mod`,

[4]Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux
[5]RyuBook: https://osrg.github.io/ryu-book/en/Ryubook.pdf
[6]OpenFlow v1.3 Messages and Structures: https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

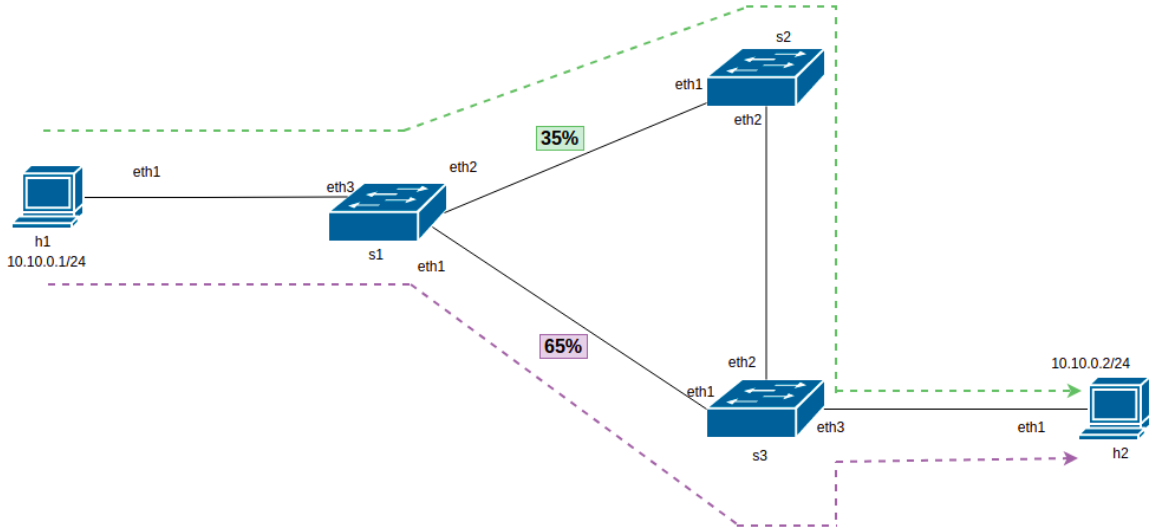


Figura A.32: Topología para división porcentual de tráfico.

`send_flow_mod`, `packet_in_handler` entre otras. Dichas funciones sirven como base y guía para el desarrollo de la aplicación.

c) Agregar la función `send_group_mod` que permite la modificación del grupo y sus tablas en el *switch* según corresponda el parámetro `datapath`. En esta función se debe definir la carga de tráfico para cada salida del grupo, en esta aplicación se definieron (`weight1=65` y `weight2=35`), así también se estructuran las acciones de salida de los `buckets`, en este caso las acciones de salida corresponden a conmutar el tráfico por el puerto 1 y 2 , definiendo las dos rutas accesibles. Se debe configurar bajo el tipo de grupo `SELECT` y definir un número para el mismo, por ejemplo `group_id=50`.

d) Incluir la función `switch_features_handler` en la aplicación de redireccionamiento, con el respectivo decorador, que permita añadir los flujos de la *table-miss*, es decir, las entradas de prioridad 0.

e) En la misma función, mediante un condicional, se deben inicializar las tablas de grupo de los *switches* extremos (`s1` y `s3`) llamando a la función `send_group_mod` para luego instalar entradas de flujo con acciones que apunten al grupo creado cuando el paquete ingresa por `eth3` en ambos *switches*. Es decir, lo que ingrese por el puerto `eth3` se reparte a través de `eth1` y `eth2` en función de los pesos asignados (`weight1=65` y `weight2=35`). Adicionalmente se deben agregar entradas de flujo que instalen rutas en sentido contrario, es decir, lo que entra por `eth1` o `eth2` sale por `eth3`.

f) Incluir la función `add_flow` con los parámetros `datapath, priority, match, actions`, esta función permite instalar flujos en los diferentes *switches*.

g) Programar la función `_packet_in_handler` con su decorador respectivo que permita procesar, extraer cabeceras, de los paquetes entrantes; mediante condicionales se debe:

- Descartar los mensajes LLDP, esto evita el reenvío del tráfico LLDP puesto que solo el controlador tiene permitido inundar paquetes, es así que se debe descartar aquellos coincidente con el tipo LLDP para que el procesamiento se detenga antes de que se pueda realizar la acción de inundar los puertos con estos paquetes.

- Aprender direcciones MAC origen/destino y relacionar con puertos correspondientes.
- Procesar paquetes *packet-in*, e instalar los flujos en los *switches* que no contengan flujos iniciales en este caso `s2`, se debe estructurar el campo de coincidencia `match` que contiene puerto de entrada, dirección ethernet origen y dirección ethernet destino; para agregar el flujo en el *switch* invocar a la función `add_flow`.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá generar tráfico de prueba en los *hosts* `h1` y `h2` utilizando la herramienta iPerf.

A.4.5. Verificación de funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Ejecutar el controlador con la aplicación desarrollada.
- Verificar entradas de flujo iniciales en los *switches*.
- Generación de tráfico para prueba de división porcentual de tráfico.
- Verificación de división porcentual de tráfico.

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `sudo python topologia_35_65.py`. Para comprobar los enlaces y elementos de la topología, se puede digitar `net`, su salida se muestra en la Figura A.33, que evidencia los enlaces entre los elementos (*switches* y *hosts*) con los puertos involucrados.

```
mininet> net
h1 h1-eth1:s1-eth3
h2 h2-eth1:s3-eth3
s1 lo:  s1-eth1:s3-eth1 s1-eth2:s2-eth1 s1-eth3:h1-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s3-eth2
s3 lo:  s3-eth1:s1-eth1 s3-eth2:s2-eth2 s3-eth3:h2-eth1
c0
```

Figura A.33: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `ryu-manager division_porcentual.py`. De manera continua se observarán registros correspondientes a los `packet in` o errores que pueda presentar la aplicación. En la Figura A.34 se puede ver una ejecución limpia, sin errores, de la aplicación `division_porcentual.py` y los primeros registros de paquetes entrantes.
3. Desde la terminal `xterm` de cualquier *switch*, se verifican las tablas de flujo iniciales, la salida debe evidenciar 4 flujos instalados en cada *switch* extremo, `s1` y `s3`, esto se muestra en la Figura A.35 marcado de rojo, en la figura se observa que tanto `s1` como `s3` tienen entradas de flujos que apuntan al grupo 50. Mientras que, `s2` únicamente tienen una entrada de flujo que corresponde a la *table-miss*, marcado de azul en la misma Figura A.35.

```
jhoss@jhossPC:~/Practicas/P4$ ryu-manager division_porcentual.py
loading app division_porcentual.py
loading app ryu.controller.ofp_handler
instantiating app division_porcentual.py of Aplicacion1
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 2 66:83:0f:35:09:df 33:33:00:00:00:fb 2
packet in 2 9a:8d:db:74:f2:21 33:33:00:00:00:fb 1
```

Figura A.34: Ejecución de la aplicación *division_porcentual.py* con el controlador Ryu y registros *packet-in*

```
root@jhossPC:~/Practicas/P4# ovs-ofctl -O openflow13 dump-flows  s1
 cookie=0x0, duration=112.508s, table=0, n_packets=0, n_bytes=0, priority=10,in_port="s1-eth3" actions=group:50
 cookie=0x0, duration=112.508s, table=0, n_packets=1, n_bytes=180, priority=10,in_port="s1-eth1" actions=output:"
s1-eth3"
 cookie=0x0, duration=112.508s, table=0, n_packets=2, n_bytes=360, priority=10,in_port="s1-eth2" actions=output:"
s1-eth3"
 cookie=0x0, duration=112.509s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
root@jhossPC:~/Practicas/P4# ovs-ofctl -O openflow13 dump-flows  s2
 cookie=0x0, duration=114.674s, table=0, n_packets=2, n_bytes=360, priority=0 actions=CONTROLLER:65535
root@jhossPC:~/Practicas/P4# ovs-ofctl -O openflow13 dump-flows  s3
 cookie=0x0, duration=116.227s, table=0, n_packets=0, n_bytes=0, priority=10,in_port="s3-eth3" actions=group:50
 cookie=0x0, duration=116.227s, table=0, n_packets=1, n_bytes=180, priority=10,in_port="s3-eth1" actions=output:"
s3-eth3"
 cookie=0x0, duration=116.227s, table=0, n_packets=2, n_bytes=360, priority=10,in_port="s3-eth2" actions=output:"
s3-eth3"
 cookie=0x0, duration=116.227s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
root@jhossPC:~/Practicas/P4#
```

Figura A.35: Tablas de flujo iniciales en cada *switch*

4. **Generación de tráfico**: Abrir la terminal `xterm` de `h1` donde se ejecutará el servidor de la herramienta iPerf mediante el comando `iperf -s`, mientras que en una terminal `xterm` de `h2` se debe ejecutar un cliente usando el comando `iperf -c 10.10.0.1 -P 160`; se usa el parámetro $-P$ que permite crear varios clientes en paralelo, lo que permite obtener un resultado más certero debido a la cantidad de tráfico; esto se indica en la Figura A.36

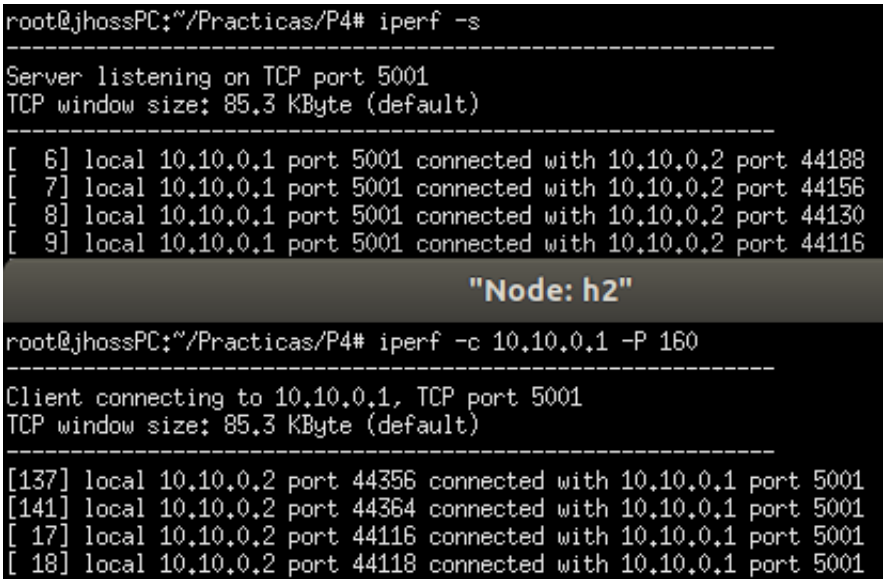


Figura A.36: Generación de tráfico con clientes en paralelo

5. **Prueba de división porcentual de tráfico**: En este paso se observan nuevamente los flujos en los *switches* que componen la red. Esto se evidencia en la Figura A.37, que muestra la agregación de dos entradas de flujo (bidireccionales) en `s2` para cursar tráfico. Para verificar la división porcentual según los pesos establecidos en el grupo se procede a ejecutar la línea `ovs-ofctl -O openflow13 dump-ports s1` que permite visualizar el tráfico cursado por los puertos del `switch 1` entonces se puede observar en la Figura A.38 que los paquetes recibidos en el puerto 1 (`rx pkts=1547535`) marcado en el recuadro rojo corresponden al 66,456 % con respecto a los paquetes transmitidos en el puerto 3 (`tx pkts=2328651`) recuadro azul, esto se debe a que el peso asignado a la salida por el puerto 1 es de `65`. Los paquetes recibidos en el puerto 2 (`rx pkts=781132`) marcado en el recuadro verde corresponden al 33,544 % de los paquetes transmitidos en el puerto 3 puesto a que el peso asignado a la salida por el puerto 2 es de `35`; como se indica los resultados son muy similares a los pesos asignados; la leve diferencia se debe a los múltiples mensajes que se dan entre *switch* y controlador, como *packet-in*, entre otros. Con lo indicado se verifica que por `s1-s3` circula el `65%` del tráfico total y por `s1-s2-s3` circula el `35%`.

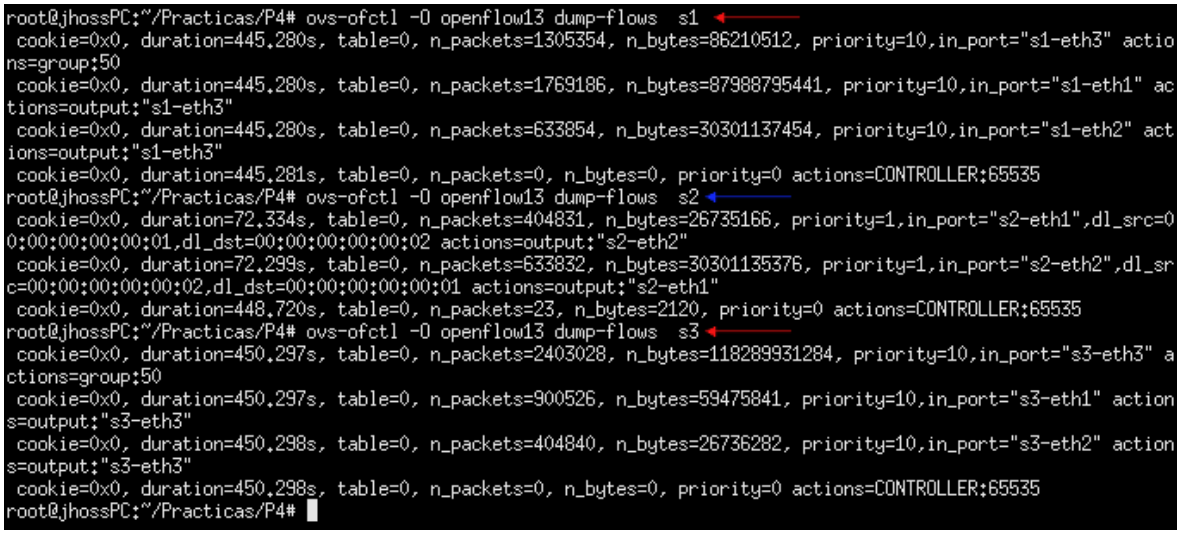


Figura A.37: Verificación de entradas de flujo instaladas para UDP y el tráfico cursado por s4

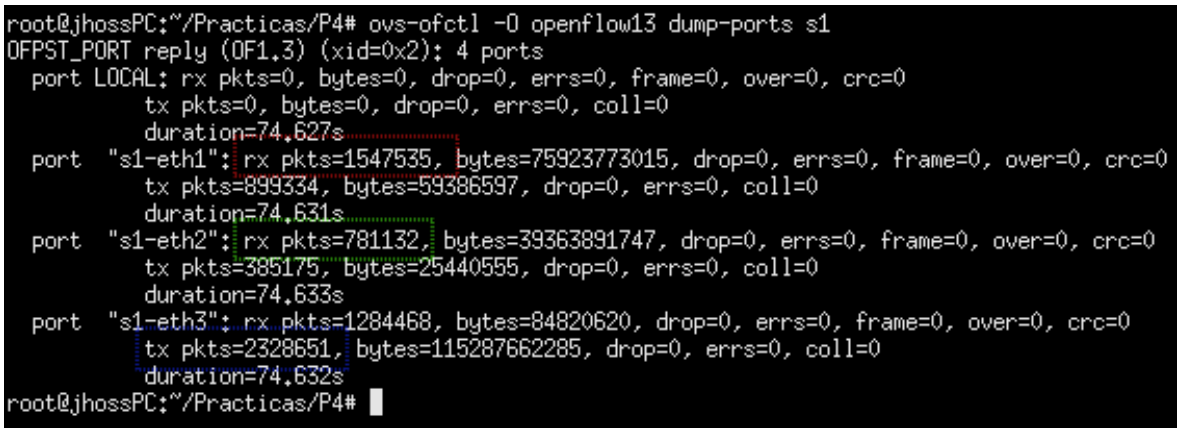


Figura A.38: Verificación de división porcentual de tráfico

A.4.6. Preguntas y resoluciones:

Una vez realizada la práctica se puede responder las siguientes interrogantes:

- En base a lo desarrollado en la práctica ¿Qué otro método se puede se puede utilizar para realizar división porcentual de tráfico?
- ¿Qué otras aplicaciones se pueden desarrollar utilizando tablas de grupo?
- Dentro de la red tradicional, ¿Existe algún protocolo o configuración de red que permita realizar una división porcentual de tráfico?. En el caso de existir, descríbalo.
- ¿Cuál es la diferencia clave entre grupo SELECT y grupo ALL? Dar un ejemplo de aplicación para cada grupo.

A.4.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.4.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.5. Práctica 5: Ingeniería de tráfico con Ryu.

Manejo de tablas de flujo y de grupo para tratamiento diferenciado de flujos TCP y UDP

A.5.1. Objetivos:

- Aplicar los conceptos de ingeniería de tráfico para proveer un tratamiento diferenciado a distintos flujos de tráfico.
- Comprender el manejo de tablas de flujo y de grupo.
- Diferenciar flujos de tráfico TCP y UDP y re-direccionarlos por rutas diferentes.
- Añadir entradas de flujo de acuerdo a distintos parámetros de coincidencia por protocolo, estableciendo instrucciones para acciones de salida y de grupo.

A.5.2. Introducción:

En esta práctica se busca direccionar el tráfico por protocolo de transporte, es decir, se pretende definir dos rutas; donde una de ellas permitirá tráfico TCP y otra UDP. La topología planteada para esta práctica se visualiza en la Figura A.39. Al finalizar esta práctica el estudiante habrá configurado una ruta bidireccional que cursará tráfico TCP por los elementos de red `s1-s2-s3` y otra ruta bidireccional que cursará tráfico UDP por `s1-s4-s3`. Para el cumplimiento de los objetivos plateados se necesita implementar la topología en el simulador Mininet y desarrollar una aplicación para el controlador Ryu en la cual principalmente se emplearán tablas de flujo y de grupo.

El tratamiento diferenciado de flujos TCP y UDP en aplicaciones prácticas reales permitiría otorgar "mejores" caminos de acuerdo a la sensibilidad que podrían presentar uno u otro servicio (streaming, transferencia de archivos, vídeo bajo demanda, etc.), otorgando QoS a la red.

Una entrada de flujo puede tener una instrucción de acción de salida hacia un grupo específico para realizar métodos adicionales como equilibrio de carga, reenrutamiento rápido, entre otros. Una tabla de grupo consta de: identificador, tipo, contadores y depósitos de acciones (`buckets`); son estas acciones de grupo las que permiten que un conjunto de puertos sean representados como una sola entidad para el reenvío de paquetes.

Para determinar si un paquete coincide con el criterio de direccionamiento, en una entrada de flujo se pueden usar diferentes campos de coincidencia como: puerto de entrada, VLAN ID, dirección MAC origen o destino, dirección IP origen o destino, protocolo, bits de tipo de servicio, puerto origen TCP/UDP, puerto destino TCP/UDP, entre otros.

En esta práctica, por tanto, para definir entradas de flujo de acuerdo a su respectivo protocolo de transporte se emplearán los campos de coincidencia: protocolo, puerto TCP/UDP destino y puerto TCP/UDP origen.

Para el desarrollo de esta práctica se requiere disponer de:

- Ordenador con alguna distribución de Sistema Operativo basada en Linux, se recomienda Ubuntu en versiones LTS[1].

[1]Página oficial de descarga de Ubuntu: `https://ubuntu.com/download/desktop`

- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[2]
- Simulador Mininet.
- Controlador Ryu.
- Herramienta iPerf.

Esta guía se ha desarrollado considerando el software más actualizado, estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.5, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.5: Software utilizado para el desarrollo de la guía Práctica 5

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 20.04, kernel 5.11.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *switch*	Open vSwitch	2.13.3
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

A.5.3. Marco teórico a desarrollar:

- Tablas de flujo y su estructura.
- Tablas de grupo y su estructura.
- Herramienta iPerf: Funcionalidad y comandos.
- Principales aplicaciones que emplean TCP y UDP.

A.5.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

1. **Crear la topología:** La topología de referencia a implementar en Mininet se muestra en la Figura A.39. Para esto se debe desarrollar un *script* en Python, `topologiaTCP-UDP.py`. La topología consta de al menos 4 *switches* OpenFlow, 2 *hosts* y al menos dos rutas alternativas.
2. **Escribir la aplicación para el controlador Ryu:** Para el desarrollo de la aplicación en lenguaje Python, denominada `tcp_udp.py`, se debe tomar en cuenta las siguientes instrucciones:
   a) Revisar la aplicación `SimpleSwitch13` presentada y explicada en el Libro RYU SDN Framework[3]. La aplicación muestra la estructura básica del código en Python que permite desarrollar aplicaciones de mayor complejidad.
   b) Consultar la documentación sobre mensajes y estructuras de OpenFlow v1.3[4]. Esta documentación describe funciones como `packet_in_handler`, `switch_features_handler`, `send_group_mod` y otras, con sus parámetros involucrados como `datapath`, `weight`, `group_id`,

[2]Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux
[3]RyuBook: https://osrg.github.io/ryu-book/en/Ryubook.pdf
[4]OpenFlow v1.3 Messages and Structures: https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

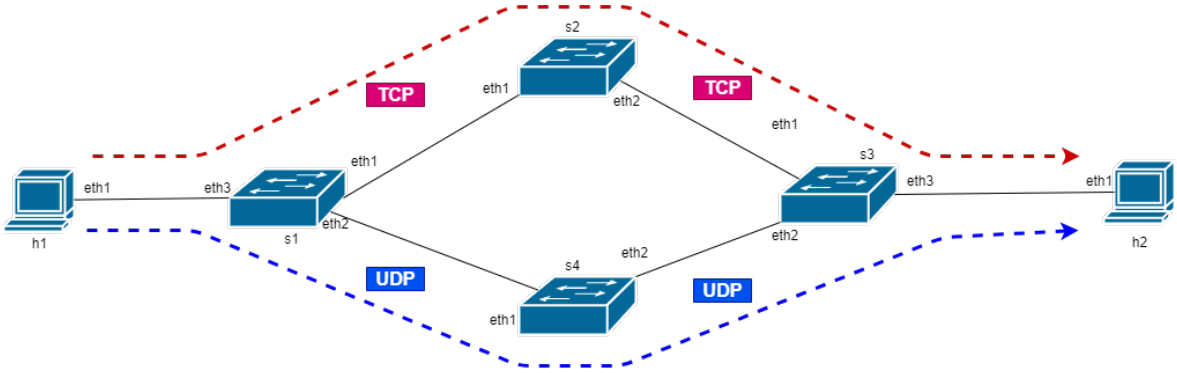


Figura A.39: Topología para direccionamiento de tráfico.

`match`, `buckets`, etc. Las funciones y parámetros mencionados se emplearán posteriormente y son imprescindibles para el desarrollo de esta aplicación.

c) Agregar la función `send_group_mod` que permite la modificación del grupo y sus tablas en el *switch* según corresponda el parámetro `datapath`. En esta función se debe definir la carga de tráfico para cada salida del grupo (`weight=100`), así también se estructuran las acciones de salida de los `buckets`. Se debe configurar bajo el tipo de grupo `ALL` y definir un número para el mismo, por ejemplo `group_id=50`.

d) Incluir la función `switch_features_handler` en la aplicación de redireccionamiento, con el respectivo decorador, que permita añadir los flujos de la *table-miss*, es decir, las entradas de prioridad 0.

e) En la misma función, mediante un condicional, se deben inicializar las tablas de grupo de los *switches* extremos (`s1` y `s3`) llamando a la función `send_group_mod` para luego instalar entradas de flujo con acciones que apunten al grupo creado cuando el paquete ingresa por `eth3`. En el caso de esta aplicación las acciones de grupo son las mismas para los dos *switches* (lo que ingrese por el puerto `eth3` se replica a través de `eth1` y `eth2`). Adicionalmente se deben agregar entradas de flujo que instalen rutas en sentido contrario, es decir, lo que entra por `eth1` o `eth2` sale por `eth3`.

f) Incluir la función `add_flow` con los parámetros `datapath, priority, match, actions`, esta función permite instalar flujos en los diferentes *switches*.

g) Programar la función `_packet_in_handler` con su decorador respectivo. Esto con el fin de que se permita procesar, y extraer las cabeceras de los paquetes entrantes. Específicamente, mediante condicionales se debe:

- Ignorar los mensajes LLDP, esto evita el reenvío del tráfico LLDP puesto que solo el controlador tiene permitido inundar paquetes, es así que se debe descartar aquellos coincidente con el tipo LLDP para que el procesamiento se detenga antes de que se pueda realiza la acción de inundar los puertos con estos paquetes.
- Aprender las direcciones MAC origen/destino y relacionarlas con los puertos correspondientes.
- Procesar los paquetes Internet Protocol version 4 (IPv4). Los paquetes ICMP servirán para crear un campo de coincidencia `match` por defecto que al instalar el flujo permite la prueba de conectividad mediante la utilidad del `ping`.

- Instalar entradas de flujo en el *switch* `s2`: si el paquete ingresa en tal *switch* y el protocolo corresponde a TCP, se debe estructurar el campo de coincidencia `match` e invocar a la función `add_flow` para agregar la entrada de flujo en dicho *switch*.
- Instalar entradas de flujo en el *switch* `s4`: si el paquete ingresa en tal *switch* y el protocolo corresponde a UDP, se debe estructurar el campo de coincidencia `match` e invocar a la función `add_flow` para agregar la entrada de flujo en el *switch*.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá generar tráfico (TCP y UDP) de prueba en los *hosts* h1 y h2.

A.5.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Ejecutar el controlador con la aplicación desarrollada.
- Verificar entradas de flujos en los *switches*.
- Generación de tráfico UDP y análisis de ruta de los flujos
- Generación de tráfico TCP y análisis de ruta de los flujos

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `$sudo python topologiaTCP-UDP.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.40, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados.

```
mininet> net
h1 h1-eth1:s1-eth3
h2 h2-eth1:s3-eth3
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s4-eth1 s1-eth3:h1-eth1
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth2 s3-eth3:h2-eth1
s4 lo:  s4-eth1:s1-eth2 s4-eth2:s3-eth2
c0
```

Figura A.40: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `ryu-manager tcp_udp.py`. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al controlador (`packet in`) o errores que pueda presentar la aplicación. En la Figura A.41 se puede ver una ejecución limpia, sin errores, de la aplicación `tcp_udp.py` y los registros de los primeros paquetes entrantes. En el caso de recibir errores, se deben resolver los mismos antes de continuar.
3. El siguiente paso consiste en verificar la correcta creación de las tablas de flujo en los *switches*. Para esto, desde la terminal `xterm` de cualquier *switch*, se verifican las tablas de flujo iniciales. La salida debe evidenciar 3 flujos instalados en cada *switch* extremo, `s1` y `s3`. Un ejemplo de esto se muestra en la Figura A.42 marcado de rojo. Por otro lado, los *switches* `s2` y `s4` únicamente

```
jofnar@jofnar-HP:~/Prácticas/P5$ ryu-manager tcp_udp.py
loading app tcp_udp.py
loading app ryu.controller.ofp_handler
instantiating app tcp_udp.py of App_tcp_udp
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 3
packet in 4 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 2 00:00:00:00:00:01 33:33:00:00:00:02 1
```

Figura A.41: Ejecución de la aplicación tcp_udp.py con el controlador Ryu y registros *packet in*

tienen una entrada de flujo que corresponde a la *table-miss* (entradas de flujo con prioridad 0), como se puede observar marcado de azul en la misma Figura A.42.

```
root@jofnar-HP:/home/jofnar/Prácticas/P5# ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=6073.632s, table=0, n_packets=32, n_bytes=4872, priority=10,in_port="s1-eth1" actions=output:"
s1-eth3"
 cookie=0x0, duration=6073.632s, table=0, n_packets=32, n_bytes=4872, priority=10,in_port="s1-eth2" actions=output:"
s1-eth3"
 cookie=0x0, duration=6073.632s, table=0, n_packets=8, n_bytes=560, priority=0 actions=CONTROLLER:65535
root@jofnar-HP:/home/jofnar/Prácticas/P5# ovs-ofctl -O OpenFlow13 dump-flows s3
 cookie=0x0, duration=6076.008s, table=0, n_packets=32, n_bytes=4872, priority=10,in_port="s3-eth1" actions=output:"
s3-eth3"
 cookie=0x0, duration=6076.008s, table=0, n_packets=32, n_bytes=4872, priority=10,in_port="s3-eth2" actions=output:"
s3-eth3"
 cookie=0x0, duration=6076.008s, table=0, n_packets=8, n_bytes=560, priority=0 actions=CONTROLLER:65535
root@jofnar-HP:/home/jofnar/Prácticas/P5# ovs-ofctl -O OpenFlow13 dump-flows s2
 cookie=0x0, duration=6088.333s, table=0, n_packets=40, n_bytes=5432, priority=0 actions=CONTROLLER:65535
root@jofnar-HP:/home/jofnar/Prácticas/P5# ovs-ofctl -O OpenFlow13 dump-flows s4
 cookie=0x0, duration=6091.919s, table=0, n_packets=40, n_bytes=5432, priority=0 actions=CONTROLLER:65535
root@jofnar-HP:/home/jofnar/Prácticas/P5#
```

Figura A.42: Tablas de flujo iniciales en cada *switch*

En este punto, el estudiante deberá ahondar en la explicación de las tablas de flujo esperadas y obtenidas para la cumplir con los objetivos la presente práctica.

4. **Generación de tráfico UDP y análisis de ruta de los flujos.**: Abrir la terminal `xterm` de `h1` donde se ejecutará el servidor de la herramienta `iperf` mediante el comando `iperf -s -u`, mientras que en una terminal `xterm` de `h2` se debe ejecutar un cliente usando el comando `iperf -c 10.10.0.1 -u`, que generará tráfico UDP; posterior a ello se deben verificar comparativamente las tablas de flujo en `s2` y `s4`. Esto se evidencia en la Figura A.43, que muestra la agregación de dos entradas de flujo (bidireccionales) en `s4` para tráfico UDP y que corresponden a los puertos utilizados en la prueba, mientras que en `s2` no se verifican cambios en las tablas puesto que este *switch* está destinado para TCP, corroborando que el tráfico UDP sigue la ruta `s1-s4-s3`. De manera adicional en la Figura A.43 se puede corroborar que la cantidad de tráfico UDP transferido (1,25 MBytes) es aproximadamente igual a la cantidad de bytes cursados por la ruta instalada (`n_bytes=1347192`≈ 1,28MBytes).

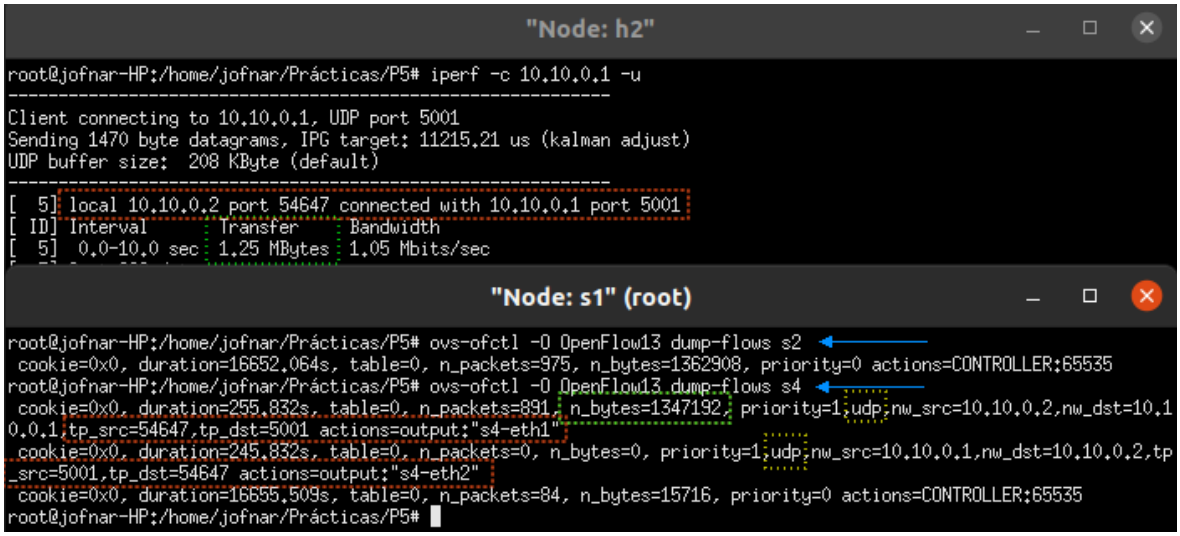


Figura A.43: Verificación de entradas de flujo instaladas para UDP y el tráfico cursado por s4

5. **Análisis con flujos de tráfico TCP**: De manera similar a la prueba para UDP, en la terminal de `h1`, el servidor de `iperf` se ejecuta con el comando `iperf -s`; mientras que en la terminal de `h2` el cliente arranca con el comando `iperf -c 10.10.0.1`, que generará tráfico TCP; nuevamente se deben verificar comparativamente las tablas de flujo en `s2` y `s4`. En la Figura A.44 se puede notar que se han agregado dos entradas de flujo (bidireccionales) en `s2` para tráfico TCP y corresponden a los puertos usados en la prueba. Al contrario en `s4` las tablas no cambian, se mantienen las entradas de flujo creadas en la prueba anterior para tráfico UDP, lo que permite corroborar que el tráfico TCP sigue la ruta `s1-s2-s3`. El tráfico TCP transferido es 14,6 MBytes que se puede verificar que han cursado desde `h2` a `h1` gracias a la ruta instalada en `s2` que registra un valor similar (`n_bytes=16371264`≈ 15,61 MBytes), así también, el enlace de sentido contrario (de `h1` a `h2`) registra el valor `n_bytes=1390268`≈ 1,33 MBytes, que corresponde a los mensajes recíprocos de retroalimentación de TCP, principalmente ACKNOWLEDGMENTs (ACKs).

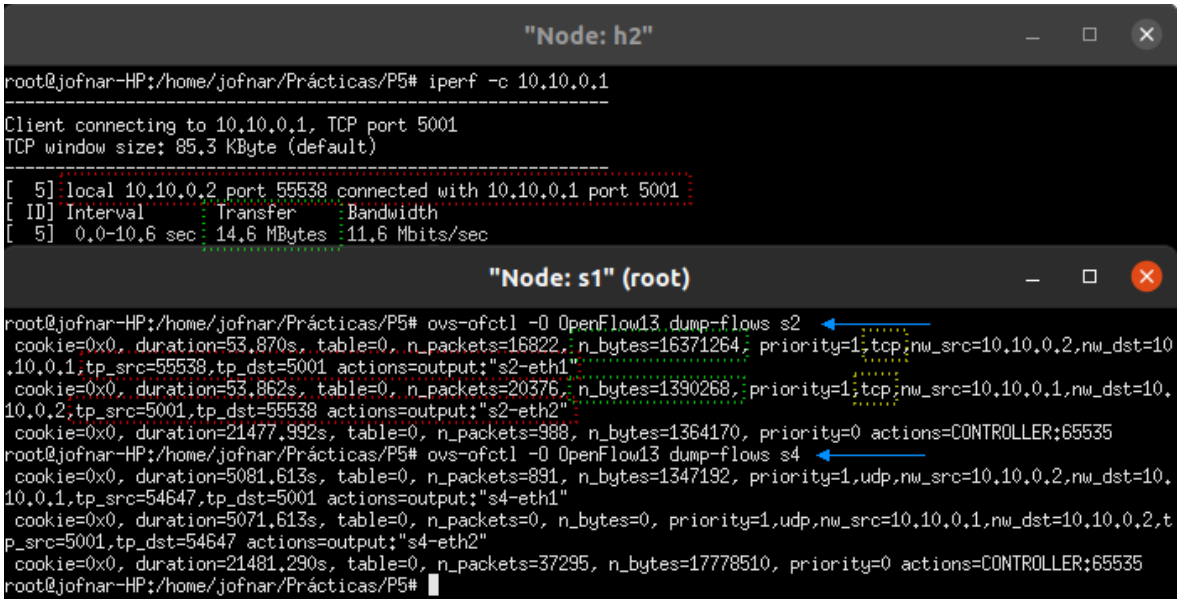


Figura A.44: Verificación de entradas de flujo instaladas para TCP y el tráfico cursado por s2

A.5.6. Preguntas y resoluciones:

Una vez realizada la práctica se puede responder las siguientes interrogantes:

- Dentro de la red tradicional, ¿Existe algún protocolo o configuración de red que permita realizar un tratamiento diferenciado de tráfico?. En el caso de existir, descríbalo.
- Desde esta experiencia, considerando otros criterios o protocolos, ¿Qué otro tratamiento diferenciado de flujos se puede configurar?
- ¿Qué tanta complejidad y utilidad cree que tiene esta implementación?
- Aparte de lo anotado en la Introducción, ¿Qué aplicaciones prácticas se le pueda dar al tratamiento diferenciado de flujos TCP y UDP?

A.5.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.5.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.6. Práctica 6: Ingeniería de tráfico con Ryu.

Redireccionamiento de Tablas de Flujo (*Pipeline*) para clasificación de tráfico

A.6.1. Objetivos:

- Clasificar tráfico en función del protocolo.
- Redireccionar entradas hacia distintas tablas de flujo.
- Entender el funcionamiento del parámetro de salida Goto-Table.

A.6.2. Introducción:

En esta práctica se busca redireccionar el tráfico según el protocolo, haciendo uso de varias tablas de flujo (*pipeline*), es decir, el tráfico al llegar a `s1` se redirecciona según este sea `TCP o UDP`, en caso de ser `TCP` se dirige hacia el puerto 1 de `s1` para llegar a `s2` y en caso de ser UDP se dirige hacia el puerto 2 de `s1` para llegar hacia `s3`. En `s2` se vuelve a redireccionar el tráfico según sea este FTP o HTTP, en caso de ser FTP se dirige hacia `h1` por medio del puerto 2 y en caso de ser HTTP se dirige hacia `h2` por el puerto 3. En `s3` igualmente se hace una redirección de tráfico UDP según este sea RTP o RTCP, en caso de que el tráfico corresponda a RTP se dirige hacia `h3` por el puerto 2 y en caso de ser RTCP se dirige hacia `h4` por el el puerto 3. Finalmente el estudiante habrá desarrollado una aplicación que permita redirigir el tráfico según el protocolo específico que se este manejando.

El Redireccionamiento de tablas de flujo (*Pipeline*) para clasificación de tráfico en aplicaciones prácticas permite seccionar un tráfico, otorgando mayor flexibilidad a las funciones del controlador, al igual que permite dar un tratamiento diferenciado según el tipo de tráfico. Cuando un *switch* posee más de una tabla de flujo estas se organizan como una tubería en donde se ordenan ascendentemente empezando por la tabla 0, usar más de una tabla de flujo proporciona flexibilidad. La instrucción *Goto-Table* permite dirigir un paquete hacia una tabla de flujo que se encuentre más adelante en el *pipeline*, no se puede accionar un flujo con una salida hacia una tabla menor que que la primera tabla. En una tubería se inicia el procesamiento de la entrada en la tabla de flujo 0, se usan otras tablas de flujo según el resultado de coincidencia en la primera tabla. Es decir, se puede tener un flujo que mediante *Goto-Table* se pueda ir a otra tabla de flujo. En caso de existir coincidencia en una o más entradas se toma la entrada de mayor prioridad la cual es otorgada por el usuario; si no existe una coincidencia en ninguna entrada el paquete es descartado. Usar varias tablas de flujo permiten que un flujo se descomponga en varios subflujos. El uso de varias tablas de flujo reduce el procesamiento del controlador SDN al igual que el conmutador, ya que las acciones de salto de tabla pueden ser definidas una vez por el controlador y examinadas una vez por el *switch*, usar tablas canalizadas permite la eficiencia de operaciones de red. [1].

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[2].
- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[3]

[1] Libro Fundations Modern Networking: http://williamstallings.com/Network/
[2] Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop
[3] Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux

- Controlador RYU.
- Mininet
- Herramienta iPerf.

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.6, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.6: Software utilizado para el desarrollo de la guía Práctica 5

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *switch*	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

A.6.3. Marco teórico a desarrollar:

- Tipos de acciones de salida. (Actions Output).
- Funcionamiento de la instrucción Goto-table.
- Tuberías en tablas de flujo (Flow Table *Pipeline*).

A.6.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el presente procedimiento:

1. **Crear la topología:** La topología personalizada a implementar en Mininet se muestra en la Figura A.45, para lo cual se debe desarrollar un *script* en Python, `topologia_Pipeline.py`. La topología consta de 3 *switches* y 5 *hosts*.
2. **Escribir la aplicación para el controlador Ryu:** Para el desarrollo de la aplicación en lenguaje Python, denominada `Pipeline.py`, se debe tomar en cuenta las siguientes instrucciones:
   a) Incluir la función `switch_features_handler` en la aplicación de redireccionamiento, con el respectivo decorador, que permita añadir los flujos de la *table-miss*, es decir, las entradas de prioridad 0.
   b) Incluir la función `add_flow` con los parámetros (table_id, datapath, priority, match, actions) esta permite instalar flujos con acciones de salida dirigidas hacia un puerto.
   c) Incluir la función `add_flow1` con los parámetros (table_id, datapath, priority, match, actions) esta permite instalar flujos con acciones de salida direccionadas hacia otra tabla de flujo.
   d) Programar la función `_packet_in_handler` con su decorador respectivo que permita procesar, extraer cabeceras, de los paquetes entrantes; mediante condicionales se debe:
      - Ignorar los mensajes LLDP, esto evita el reenvío del tráfico LLDP puesto que solo el controlador tiene permitido inundar paquetes, es así que se debe descartar aquellos

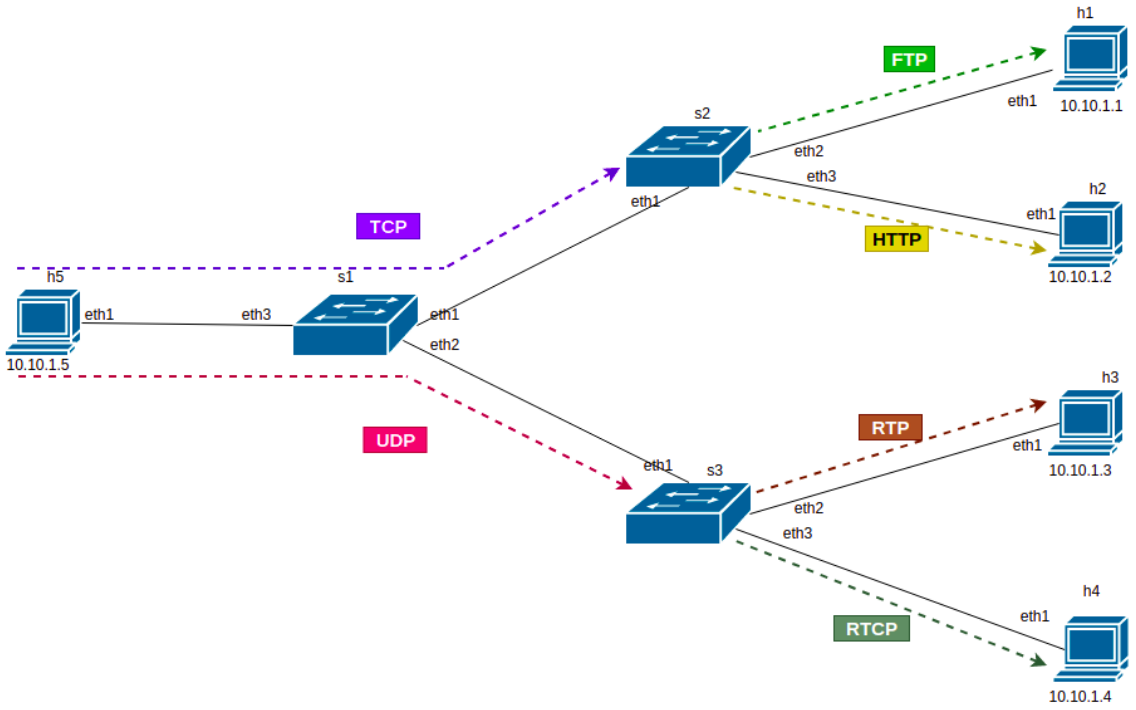


Figura A.45: Topología para *Pipeline* para clasificación de tráfico.

coincidente con el tipo LLDP para que el procesamiento se detenga antes de que se pueda realiza la acción de inundar los puertos con estos paquetes.

- Aprender direcciones MAC origen/destino y relacionar con puertos correspondientes.
- Procesar paquetes IPv4, los paquetes ICMP servirán para crear un campo de coincidencia `match` que al instalar el flujo permite la prueba de conectividad mediante `ping`.
- Instalar los flujos en `s1`: Este *switch* contendrá 3 tablas de flujo. En la tabla de flujo 0 se encontrarán entradas de flujo, con las siguientes acciones de salida, adicional a las entradas de prioridad 0:
  - Si el protocolo corresponde a TCP, se define una acción de salida que apunta a la tabla de flujo `1`.
  - Si el protocolo corresponde a UDP, se define una acción de salida que apunta a la tabla de flujo `2`.

  En la tabla de flujo 1 se tendrán entradas de flujo con las siguientes acciones de salida:
  - Si el protocolo corresponde a TCP y la dirección fuente es la de h5 (10.10.1.5) se define una acción de salida hacia el puerto `1`, en caso de otra dirección fuente se define una acción de salida hacia el puerto `3`.

  En la tabla de flujo 2 se tendrán entradas de flujo con las siguientes acciones de salida:
  - Si el protocolo corresponde a UDP y la dirección fuente es la de h5 (10.10.1.5) se define una acción de salida hacia el puerto `2`, en caso de otra dirección fuente se define una acción de salida hacia el puerto `3`.
- Instalar los flujos en `s2`: Este *switch* contendrá 3 tablas de flujo. En la tabla de flujo 0 se encontraran entradas de flujo, con las siguientes acciones de salida, adicional a las

entradas de prioridad 0:

- Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto TCP origen es `20 o 21`, se define una acción de salida que apunta a la tabla de flujo `1`.
- Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto TCP origen es `80`, se define una acción de salida que apunta a la tabla de flujo `2`.
- Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto TCP destino es `20 o 21`, se define una acción de salida que apunta a la tabla de flujo `1`.
- Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto TCP destino es `80`, se define una acción de salida que apunta a la tabla de flujo `2`.

En la tabla de flujo 1 se tendrán entradas de flujo con las siguientes acciones de salida:

- Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto TCP origen es `20 o 21`, se define una acción de salida que apunta al puerto `2`.
- Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto TCP destino es `20 o 21`, se define una acción de salida que apunta al puerto `1`.

En la tabla de flujo 2 se tendrán entradas de flujo con las siguientes acciones de salida:

- Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto TCP origen es `80`, se define una acción de salida que apunta al puerto `3`.
- Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto TCP destino es `80`, se define una acción de salida que apunta al puerto `1`.

- Instalar los flujos en `s3`: Este *switch* contendrá 3 tablas de flujo. En la tabla de flujo 0 se encontrarán entradas de flujo, con las siguientes acciones de salida, adicional a las entradas de prioridad 0:
  - Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto UDP origen es `1024`, se define una acción de salida que apunta a la tabla de flujo `1`.
  - Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto UDP origen es `1025`, se define una acción de salida que apunta a la tabla de flujo `2`.
  - Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto UDP destino es `1024`, se define una acción de salida que apunta a la tabla de flujo `1`.
  - Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto UDP destino es `1025`, se define una acción de salida que apunta a la tabla de flujo `2`.

  En la tabla de flujo 1 se tendrán entradas de flujo con las siguientes acciones de salida:

  - Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto UDP origen es `1024`, se define una acción de salida que apunta al puerto `2`.
  - Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto UDP destino es `1024`, se define una acción de salida que apunta al puerto `1`.

  En la tabla de flujo 2 se tendrán entradas de flujo con las siguientes acciones de salida:

  - Si la dirección IP origen es la de h5 (10.10.1.5) y el puerto UDP origen es `1025`, se define una acción de salida que apunta al puerto `3`.
  - Si la dirección IP origen no es la de h5 (10.10.1.5) y el puerto UDP destino es `1025`, se define una acción de salida que apunta al puerto `1`.

  Para instalar los flujos se estructura el campo match y se invoca a la función add_flow o add_flow1 dependiendo de la acción de salida.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá generar tráfico de prueba en los *hosts* h1 y h2, el tráfico generado debe ser (TCP o UDP) considerando los puertos respectivos para cada protocolo.

A.6.5. Verificación de funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

1. Ejecutar la topología creada.
2. Ejecutar el controlador con la aplicación desarrollada.
3. Verificar entradas de flujo iniciales en los *switches*.
4. Generar tráfico TCP a través del puerto 21 (FTP) y verificar las tablas y entradas de flujo adicionadas.
5. Generar tráfico TCP a través del puerto 80 (HTTP) y verificar las tablas y entradas de flujo adicionadas.
6. Generar tráfico UDP a través del puerto 1024 (RTP) y verificar las tablas y entradas de flujo adicionadas.
7. Generar tráfico UDP a través del puerto 1025 (RTCP) y verificar las tablas y entradas de flujo adicionadas.
8. Análisis de ruta de los flujos.

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `sudo python topologia_Pipeline.py`. Para comprobar los enlaces y elementos de la topología, se puede digitar `net`, su salida se muestra en la Figura A.46 que evidencia los enlaces entre los elementos (*switches* y *hosts*) con los puertos involucrados.

```
mininet> net
h1 h1-eth1:s2-eth2
h2 h2-eth1:s2-eth3
h3 h3-eth1:s3-eth2
h4 h4-eth1:s3-eth3
h5 h5-eth1:s1-eth3
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s3-eth1 s1-eth3:h5-eth1
s2 lo:  s2-eth1:s1-eth1 s2-eth2:h1-eth1 s2-eth3:h2-eth1
s3 lo:  s3-eth1:s1-eth2 s3-eth2:h3-eth1 s3-eth3:h4-eth1
c0
```

Figura A.46: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `ryu-manager Pipeline.py`. De manera continua se observarán registros correspondientes a los `packet in` o errores que pueda presentar la aplicación. En la Figura A.47 se puede ver una ejecución limpia, sin errores, de la aplicación `Pipeline.py` y los primeros registros de paquetes entrantes. En caso de obtener algún error, realizar la respectiva corrección antes de continuar.

```
jhoss@jhossPC:~/Practicas/P6$ ryu-manager Pipeline.py
loading app Pipeline.py
loading app ryu.controller.ofp_handler
instantiating app Pipeline.py of Pipeline
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 c6:f7:87:94:9d:42 33:33:00:00:00:02 1
packet in 3 c6:f7:87:94:9d:42 33:33:00:00:00:02 1
packet in 3 00:00:00:00:00:03 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 2
packet in 2 00:00:00:00:00:03 33:33:00:00:00:02 1
packet in 1 fe:ef:cd:46:e1:12 33:33:00:00:00:02 2
```

Figura A.47: Ejecución de la aplicación *Pipeline.py* con el controlador Ryu y registros *packet in*

3. Desde la terminal `xterm` de cualquier *switch*, se verifican las tablas de flujo iniciales, la salida debe evidenciar 1 una entrada de flujo en cada *switch* que corresponde a la *table-miss*, entradas de flujo con prioridad cero. Esto se evidencia en la Figura A.48, marcado de verde, azul y rojo, para s1, s2 y s3 respectivamente.

```
root@jhossPC:~/Practicas/P6# ovs-ofctl -O openflow13 dump-flows s1
 cookie=0x0, duration=80.738s, table=0, n_packets=18, n_bytes=1792, priority=0 actions=CONTROLLER:65535
root@jhossPC:~/Practicas/P6# ovs-ofctl -O openflow13 dump-flows s2
 cookie=0x0, duration=82.279s, table=0, n_packets=18, n_bytes=1792, priority=0 actions=CONTROLLER:65535
root@jhossPC:~/Practicas/P6# ovs-ofctl -O openflow13 dump-flows s3
 cookie=0x0, duration=83.886s, table=0, n_packets=18, n_bytes=1792, priority=0 actions=CONTROLLER:65535
root@jhossPC:~/Practicas/P6#
```

Figura A.48: Tablas de flujo iniciales en cada *switch*

4. **Generación de tráfico TCP a través del puerto 21 (FTP) y análisis de rutas**: Abrir la terminal `xterm` de `h5` donde se ejecutará el servidor de la herramienta `iperf` mediante el comando `iperf -s -p 21`, $-p$ hace referencia al puerto a través del cual se enviará el tráfico; en una terminal `xterm` de `h1` se debe ejecutar un cliente usando el comando `iperf -c 10.10.1.5 -p 21`, que generará tráfico TCP (FTP); posterior a ello se deben verificar las tablas y entradas de flujo en cada *switch*. Esto se evidencia en la Figura A.49, que muestra la agregación de dos entradas de flujo (bidireccionales) en la tabla de flujo 0 (recuadros rojos) y 2 entradas de flujo en la tabla de flujo 1 (recuadros verdes), pertenecientes a `s1;` (marcado de verde) y en `s2` (marcado de azul) se han generado igualmente dos tablas de flujo con dos entradas de flujo cada una. Mientras que en `s3` (marcado de rojo) se mantiene solo con el flujo de prioridad 0. Con esto se puede corroborar que el tráfico FTP cursa s1 y s2, también según las entradas generadas se observa en `s1` tabla 0 que si el flujo es TCP se dirige hacia la tabla de flujo 1 en la cual se indica que si el flujo es TCP sale por el `puerto 1 o 3` según la dirección IP origen, al salir por el puerto 1 este llega hacia `s2` en donde si es FTP (TCP puerto 21) se dirige hacia la tabla de flujo 1 en la cual sus entradas de flujo indican que dependiendo la dirección IP origen su salida sera por el puerto 1 o 2, al salir por el puerto 2 se dirige hacia h1. Entonces el tráfico FTP cursa la ruta bidireccional `h1-s1(eth1)-s2(eth2)-h1`.

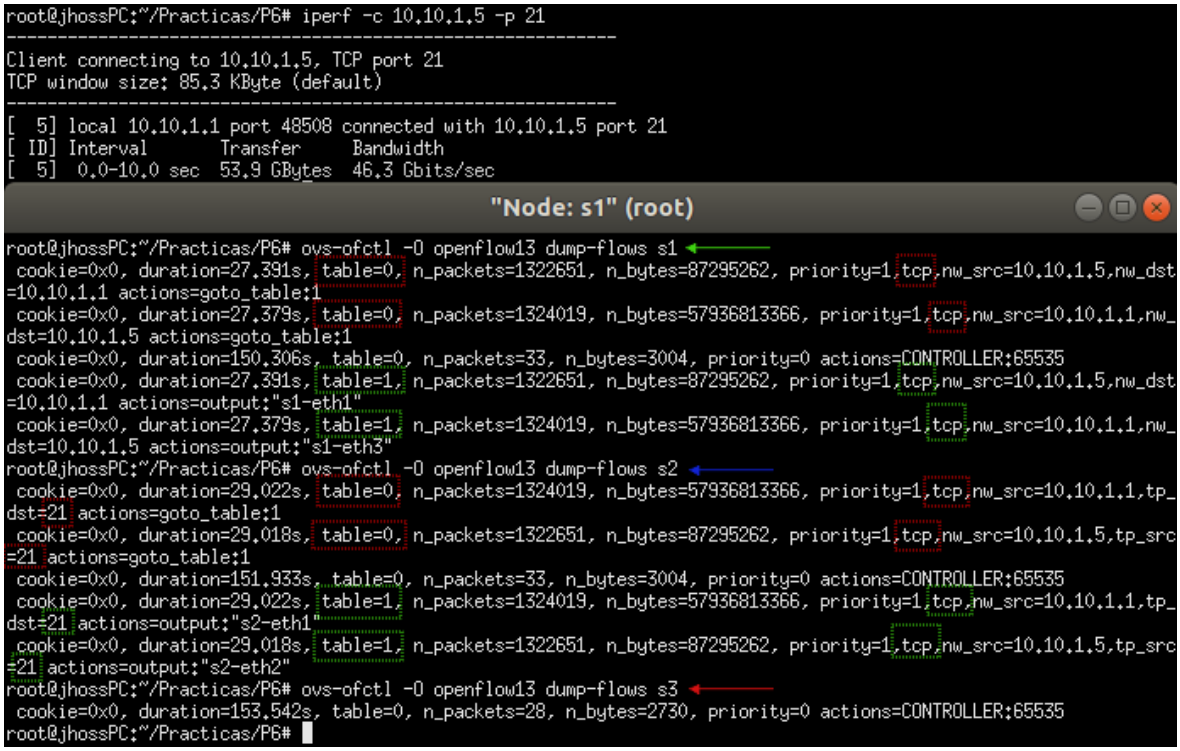


Figura A.49: Verificación de entradas de flujo instaladas para FTP y el tráfico cursado por s1 y s2

5. **Generación de tráfico TCP a través del puerto 80 (HTTP) y análisis de rutas**: El *host* `h5` actuará como servidor y `h2` sera cliente HTTP; posterior a ello se deben verificar las tablas y entradas de flujo en cada *switch*. Esto se evidencia en la Figura A.50, que muestra en `s1` la agregación de dos entradas de flujo (bidireccionales) en la tabla de flujo 0 (recuadros rojos) y 2 entradas de flujo en la tabla de flujo 1 (recuadros verdes), en `s2` (marcado de azul) se han agregado igualmente dos entradas de flujo en la tabla 0 (recuadros rojos) y 2 entradas de flujo en la tabla 2. Con esto se puede corroborar que el tráfico HTTP cursa s1 y s2, también según las entradas generadas se observa en `s1` tabla 0 que si el flujo es TCP se dirige hacia la tabla 1 en la cual se indica que si el flujo es TCP sale por el `puerto 1 o 3` según la dirección IP origen, al salir por el puerto 1 este llega hacia `s2` (tabla 0) en donde si es HTTP (TCP puerto 80) se dirige hacia la tabla de flujo 2 en la cual sus entradas de flujo indican que dependiendo la dirección IP origen su salida sera por el puerto 1 o 3, al salir por el puerto 3 se dirige hacia h2. Entonces el tráfico HTTP cursa la ruta bidireccional `h1-s1(eth1)-s2(eth3)-h2`. Pues en `s3` no se han agregado tablas o entradas de flujo aparte de la entrada de flujo de prioridad 0.

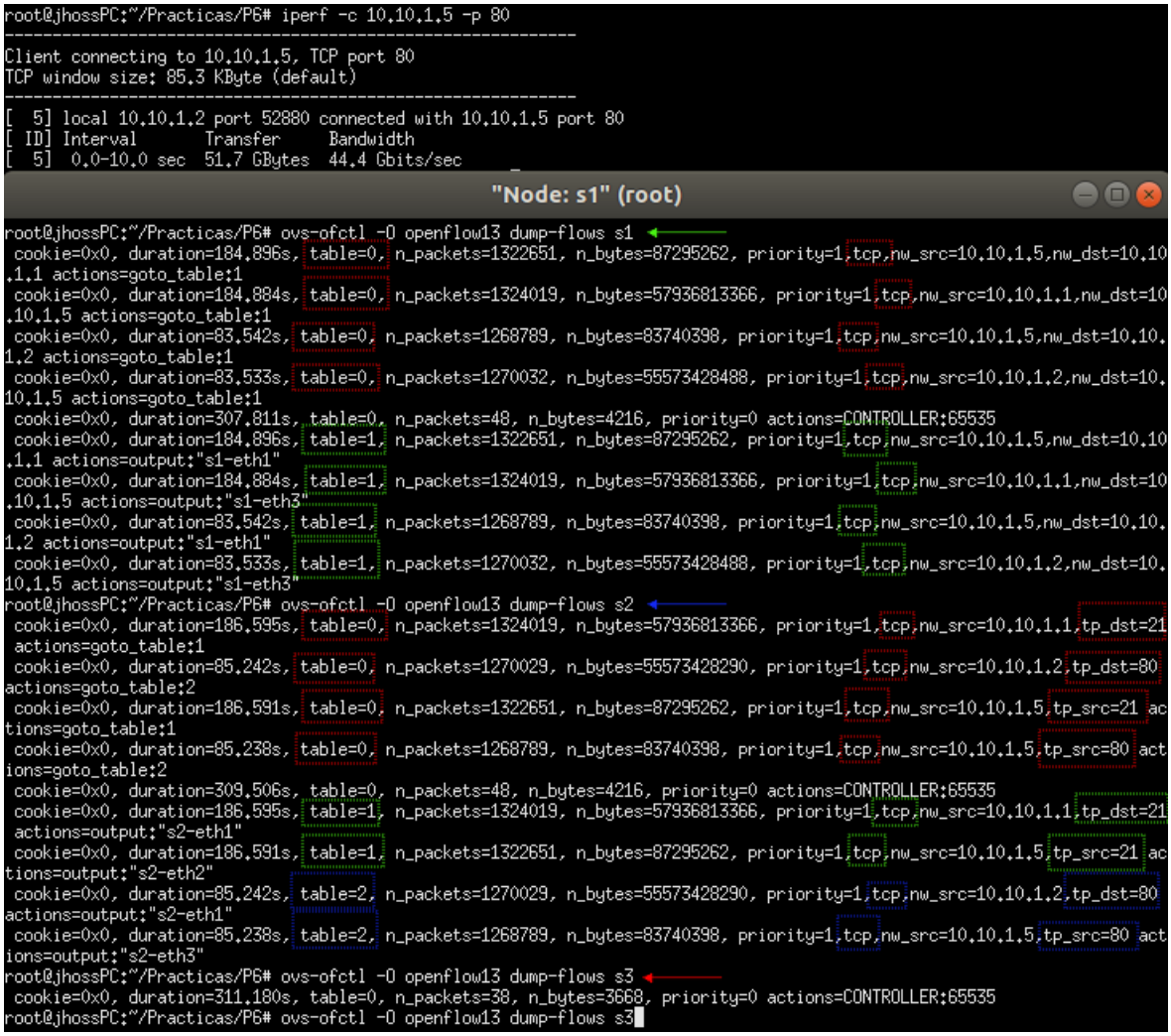


Figura A.50: Verificación de entradas de flujo instaladas para HTTP y el tráfico cursado por s1 y s2

6. **Generación de tráfico UDP a través del puerto 1024 (RTP) y análisis de rutas**: El *host* `h5` actuará como servidor y `h3` será cliente RTP; posterior a ello se deben verificar las tablas y entradas de flujo en cada *switch*. Esto se evidencia en la Figura A.51, que muestra en `s1` la agregación de dos entradas de flujo (bidireccionales) en la tabla de flujo 0 (recuadros rojos) y 2 entradas de flujo en la tabla de flujo 2 (recuadros azules), mientras que las entradas de flujo de la tabla de flujo 1 se mantienen constantes. En `s2` (marcado de azul) las entradas de flujo tanto de la tabla de flujo 0 como de la tabla de flujo 1 y 2 se mantienen constantes lo que indica que por s2 no circula tráfico RTP. En `s3` se han creado dos tablas de flujo; la tabla de flujo 0 (recuadros rojos) y la tabla de flujo 1 (recuadros verdes) cada una con dos entradas de flujo. Según las entradas de flujos agregadas la ruta del tráfico RTP es la siguiente `h1-s1(eth2)-s3(eth2)-h3`.

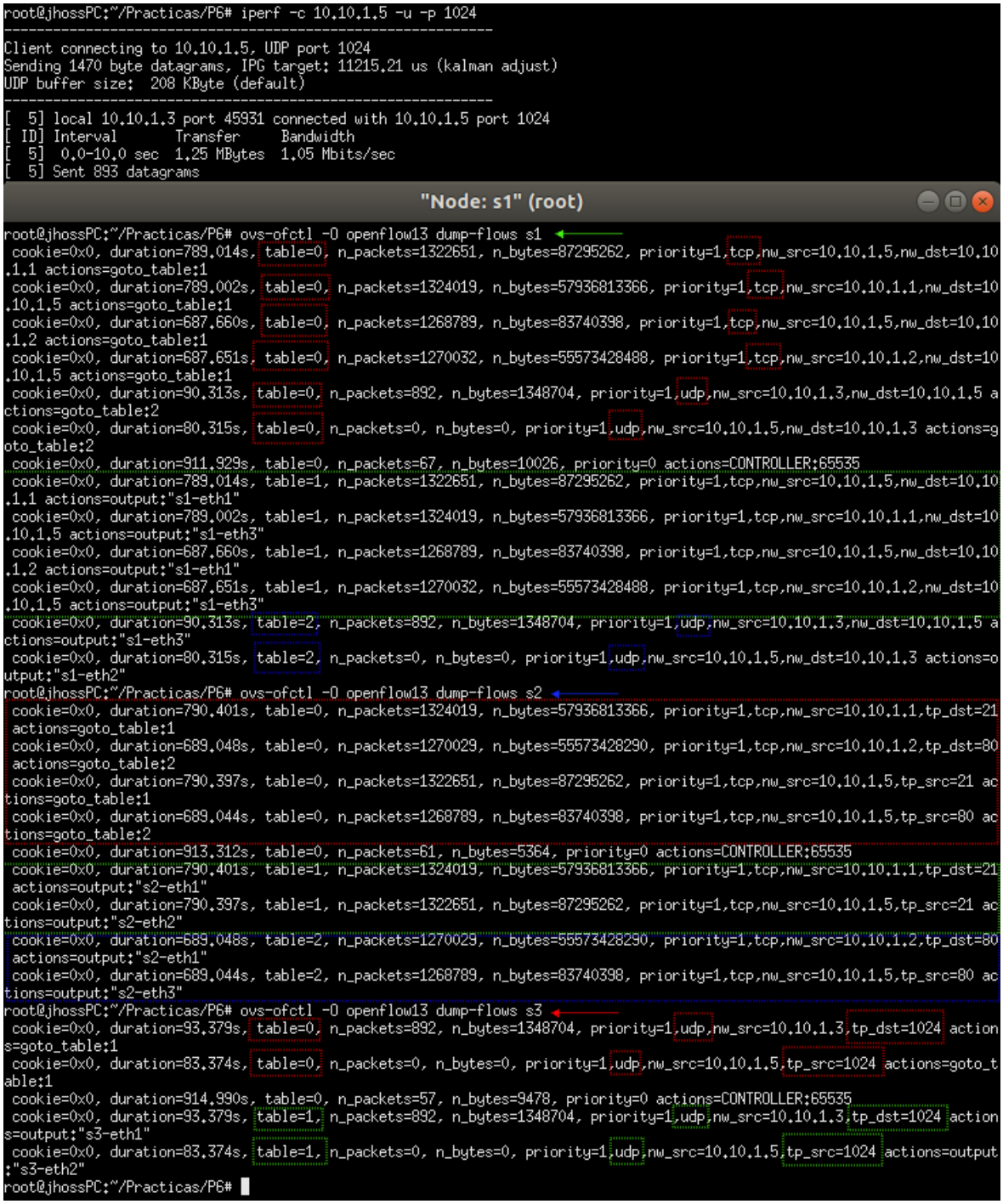


Figura A.51: Verificación de entradas de flujo instaladas para RTP y el tráfico cursado por s1 y s3

7. **Generación de tráfico UDP a través del puerto 1025 (RTCP) y análisis de rutas**: El *host* `h5` actuará como servidor y `h4` será cliente RTCP; posterior a ello se deben verificar las tablas y entradas de flujo en cada *switch*. Esto se evidencia en la Figura A.52, que muestra en `s1` la agregación de dos entradas de flujo (bidireccionales) en la tabla de flujo 0 (recuadros rojos) y 2 entradas de flujo en la tabla de flujo 2 (recuadros azules), mientras que las entradas de flujo de la tabla de flujo 1 se mantienen constantes. En `s2` (marcado de azul) las entradas de flujo tanto

de la tabla de flujo 0 como de la tabla de flujo 1 y 2 se mantienen constantes lo que indica que por `s2` no circula tráfico RTCP. En `s3` se han agregado 2 entradas de flujo a la tabla de flujo 0 y dos entradas de flujo en la tabla de flujo 2 mientras que las entradas de flujo en la tabla de flujo 1 se mantienen constantes. Según las entradas de flujos agregadas la ruta del tráfico RTP es la siguiente `h1-s1(eth2)-s3(eth3)-h4`.

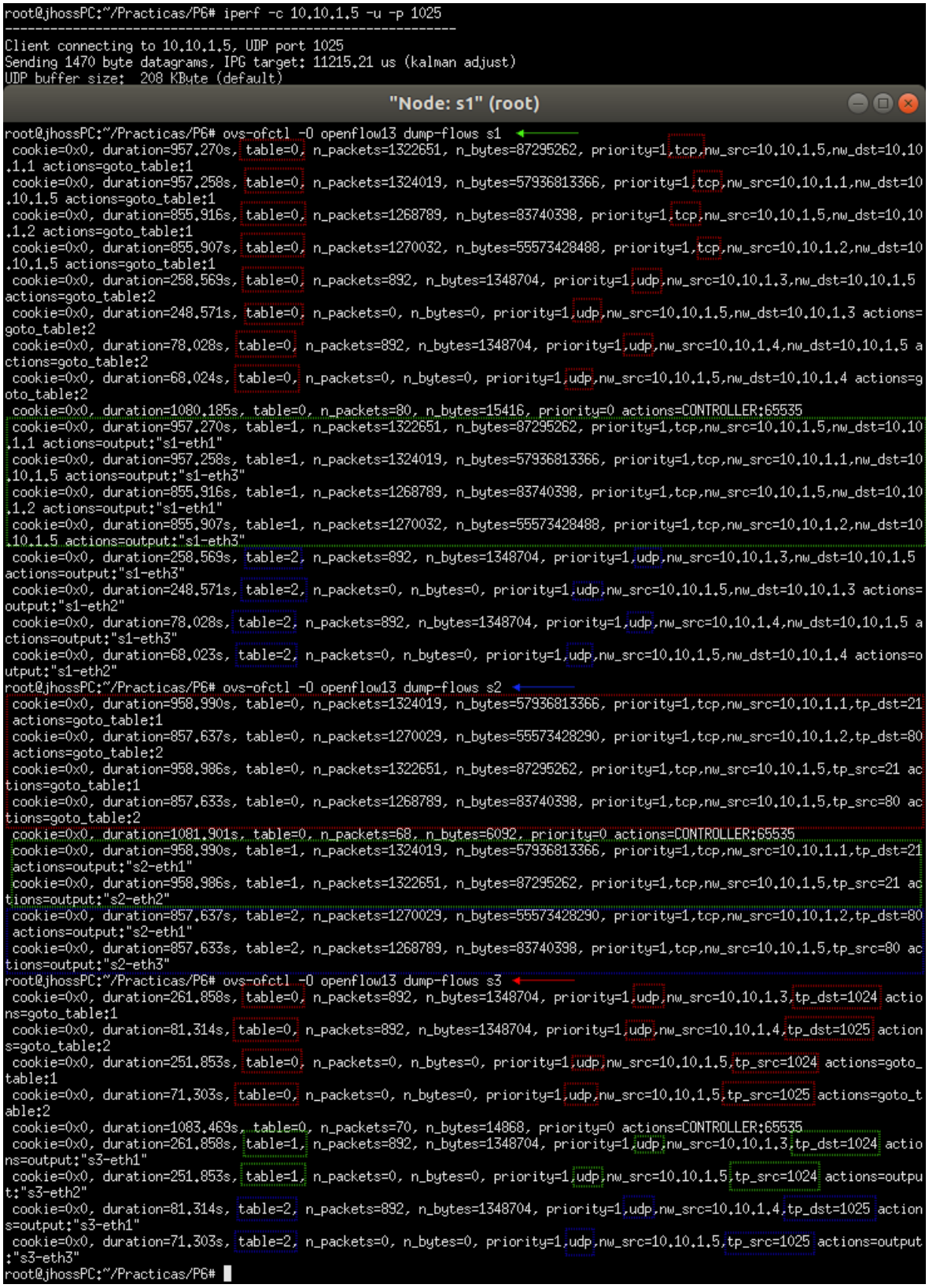


Figura A.52: Verificación de entradas de flujo instaladas para RTCP y el tráfico cursado por s1 y s3

Finalmente se realizó una prueba en la cual `h5` hace de servidor UDP(RTP) y se intenta enviar tráfico RTP desde `h2`. Los resultados se evidencian en la Figura A.53 en donde los flujos antes de realizar la prueba están marcados de azul y después de la prueba (marcado de verde), se visualizan nuevamente las entradas de flujo en s2. Se puede ver que no existe cambio alguno, es decir, la cantidad de paquetes es la misma antes y después de la prueba, lo que demuestra que RTP sigue la ruta establecida `h1-s1(eth2)-s3(eth2)-h3`.

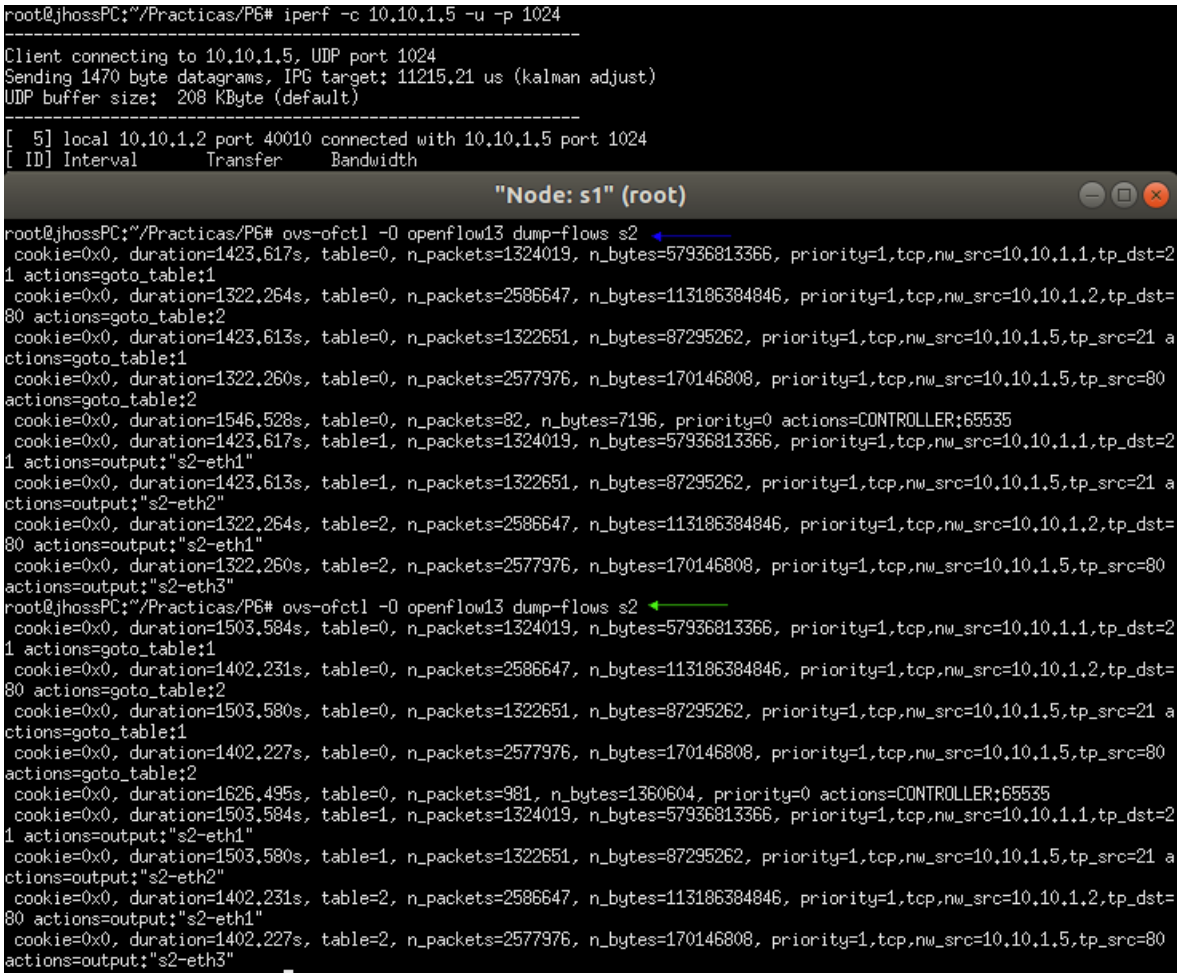


Figura A.53: Verificación de de tráfico UDP (RTP) desde h2.

A.6.6. Preguntas y resoluciones:

Una vez realizada la práctica se puede responder las siguientes interrogantes:

- ¿Cuál es la diferencia entre OFPActionOutput y OFPInstruction?
- ¿Es posible conmutar un flujo hacia una tabla de ID inferior que la tabla de ingreso? ¿En caso afirmativo que sucedería?

A.6.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.6.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.7. Práctica 7: Descubrimiento de rutas basado en ARP

Evaluación y comprensión del comportamiento de una aplicación existente

A.7.1. Objetivos:

- Evaluar el comportamiento de una aplicación existente para el descubrimiento de rutas.
- Comprender todas las funcionalidades programadas originalmente en la aplicación.
- Modificar parámetros de entrada para analizar el comportamiento ajustado a una topología diferente a la que fue planteada.
- Listar las rutas posibles desde un *host* a otro.

A.7.2. Introducción:

En esta práctica se pretende evaluar el comportamiento de una aplicación existente que, entre sus múltiples funcionalidades, realiza el descubrimiento y listado de rutas disponibles desde un *host* a otro. Esta aplicación está documentada y anexa en el trabajo "*Estudio de técnicas de Ingeniería de Tráfico basadas en SDN*" [1], que en adelante se denominará como "*documentación base*". La aplicación en mención ha sido desarrollada para el controlador Ryu con el fin de realizar un balanceo de carga por distintos caminos, basado en el costo de enlace. La topología en la que se ha evaluado es la mostrada en la Figura A.54.

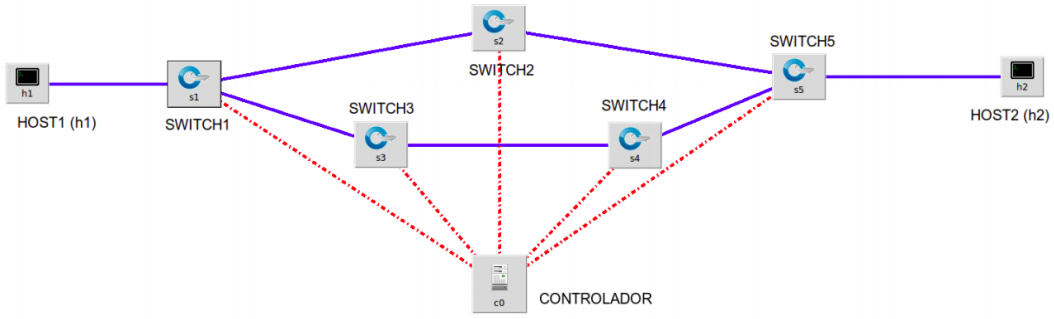


Figura A.54: Topología multicamino elegida en la Documentación base[1].

Esta topología esta conformada por 5 *switches* y 2 *hosts*, aparte del controlador. Presenta la limitación de únicamente 2 caminos disponibles para el tráfico `h1-h2`, estos caminos son `s1-s2-s5` (el más corto) y `s1-s3-s4-s5` (el más largo). Por tal motivo se propone evaluar la aplicación y extender su funcionamiento empleando la topología de la Figura A.55. La topología propuesta, cuenta con igual número de *hosts* pero menor cantidad de *switches*, sin embargo, debido a la configuración de las conexiones permite más caminos posibles entre `h1-h2`; de manera evidente y orden creciente por extensión se tienen los caminos: `s1-s4`, `s1-s3-s4` y `s1-s2-s3-s4`.

La aplicación a evaluar, según su documentación, presenta algoritmos para las funcionalidades de:

1. Descubrimiento de rutas basado en el procesamiento de mensajes ARP.
2. Listado de las rutas disponibles desde un *host* a otro.

[1]Study of SDN Traffic Engineering techniques: http://repositorio.unican.es:8080/xmlui/bitstream/handle/10902/14193/409476.pdf

3. Cálculo del coste de camino basado en el número de saltos y ancho de banda de los enlaces.
4. Instalación de flujos sobre las rutas bajo consideraciones de equilibrio de carga de acuerdo al coste de camino.
5. Estimación del tiempo total requerido para la instalación de un camino.

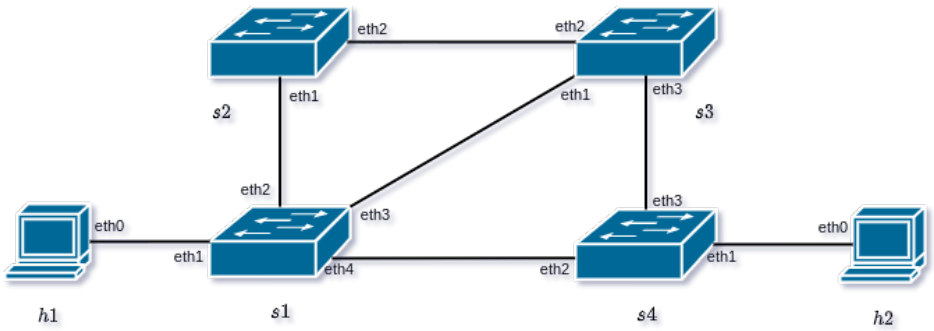


Figura A.55: Topología multicamino propuesta para esta práctica.

Las dos primeras funcionalidades se dan gracias a algoritmos de búsqueda de caminos, entre los principales se tiene la búsqueda primero por amplitud y la búsqueda primero por profundidad. Estos algoritmos de búsqueda agotan todas las posibilidades iterando todos los caminos posibles, desde el nodo origen, hasta alcanzar el nodo destino. Esta característica de búsqueda los hace diferentes respecto al Algoritmo de Dijkstra. De manera específica, la aplicación emplea la búsqueda por profundidad y recopila la información de nodos adyacentes alcanzables mediante el procesamiento de mensajes ARP.

La tercera funcionalidad, el cálculo de coste de camino, emplea dos operaciones:

- Primero el cálculo de costo por cada enlace en el camino.
- Luego, se estima el costo total por medio de la sumatoria de todos los costes de enlace involucrados en el camino.

La funcionalidad de instalación de flujos para el balanceo de carga se obtiene bajo un algoritmo que determina si los *switches* del camino son nodos de enlace de varios caminos entonces se instala flujos de tabla de grupo, tipo `SELECT`; caso contrario se instalan únicamente entradas de flujos de reenvió de paquetes. El concepto, características y funcionalidades de tablas de grupo fue abordado en prácticas anteriores, resultando familiar esta implementación para el equilibrio de carga.

La última funcionalidad estima el tiempo requerido para instalar un camino, esto se logra por instrucciones de código de menor complejidad dentro de la aplicación. Para este fin se utiliza una librería de Python especializada para el registro de tiempos desde el inicio y final del procedimiento, luego se calcula la diferencia de tiempo.

La evaluación de esta aplicación en la topología propuesta es un preámbulo de familiarización con los algoritmos y el código de la misma que permita luego, desarrollar funcionalidades esenciales para redes modernas. Una de estas funcionalidades es el reenrutamiento rápido, planteado en una práctica posterior y que tiene como fundamento la evaluación que se aborda en la presente práctica.

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[2].

[2]Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop

- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[3]
- Controlador RYU.
- Mininet

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.8, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.7: Software utilizado para el desarrollo de la guía Práctica 7

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 20.04, kernel 5.11.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *switch*	Open vSwitch	2.13.3
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

A.7.3. Marco teórico a desarrollar:

- Algoritmos de búsqueda de caminos, al menos:
  - Algoritmo de Dijkstra
  - Búsqueda primero por amplitud
  - Búsqueda primero por profundidad
- Procedimientos para el cálculo de coste de enlace y de camino.
- Tablas de grupo, especialmente tipo `SELECT`

A.7.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el procedimiento que se describe a continuación, así como, es de vital importancia leer previamente y apoyarse simultáneamente de la documentación base[1] presentada en el apartado anterior.

1. **Crear la topología:** La topología propuesta a implementar en Mininet se muestra en la Figura A.55. Para esto se debe desarrollar un *script* en Python, `topoMulticamino.py`. La topología consta de 4 *switches* y 2 *hosts*.
2. **Escribir la aplicación para el controlador Ryu:** Para el desarrollo de la aplicación en lenguaje Python, denominada `multicamino.py`, se empleará la aplicación del mismo nombre incluida en Anexos de la documentación base[1]. Esta aplicación se encuentra documentada y para adaptarla a la topología de la Figura A.55 se debe tomar en cuenta las siguientes instrucciones:

   a) Revisar el código de la aplicación `multicamino.py` presentada y explicada en la documentación base[1]. La aplicación esta completamente programada en Python y permite desplegar

[3]Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux

las funcionalidades descritas en el apartado introductorio A.7.2, entre ellas el descubrimiento de rutas.

b) Clonar (copiar o transcribir) completamente el código de `multicamino.py` en un *script* del mismo nombre.

c) Habilitar la creación de hasta 3 rutas cambiando el valor de la variable global `MAX_PATHS=2`, por defecto está indicado que se consideren solo 2 rutas.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá realizar pruebas de conectividad mediante `ping` y visualizar el resultado del descubrimiento de rutas.

A.7.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Ejecutar el controlador con la aplicación a evaluar.
- Verificar la conexión entre *hosts*.
- Analizar el descubrimiento de rutas mediante los mensajes que presenta el controlador.

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `sudo python topoMulticamino.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.56, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s4-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s3-eth1 s1-eth4:s4-eth2
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s3-eth2
s3 lo:  s3-eth1:s1-eth3 s3-eth2:s2-eth2 s3-eth3:s4-eth3
s4 lo:  s4-eth1:h2-eth0 s4-eth2:s1-eth4 s4-eth3:s3-eth3
c0
```

Figura A.56: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. Antes de ejecutar el controlador, se debe asegurar que los conmutadores empleen el protocolo OpenFlow versión 1.3. Con este fin, se abre las terminales `xterm` de cada uno de los *switches* y se ejecuta la línea de comando: `ovs-vsctl set Bridge s1 protocols=OpenFlow13`, donde `s1` corresponde al identificador del *switch*.
3. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `ryu-manager --observe-links multicamino.py`. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al controlador (`packet in`) o errores que pueda presentar la aplicación. En la Figura A.57 se puede ver una ejecución limpia, sin errores, de la aplicación `multicamino.py`. En el caso de recibir errores, se deben resolver los mismos antes de continuar.

   En la Figura A.57 se muestran los registros `Se ha llamado a switch_features_handler` que corresponden uno por cada *switch*, esto está programado dentro del evento del mismo nombre

```
jofnar@jofnar-HP:~$ ryu-manager --observe-links P7/multicamino.py
loading app P7/multicamino.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app P7/multicamino.py of ProjectController
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
```

Figura A.57: Registros de mensajes en el controlador tras la ejecución de la aplicación multicamino.py

(Manejador de Características del *switch*) que es llamado cuando un *switch* se presenta ante el controlador. La conexión del controlador a la topología crea entradas de flujo iniciales sobre cada *switch*. Un ejemplo de las entradas de flujo en `s3` se muestra en la Figura A.58 tras ejecutar la consulta por medio de `ovs-ofctl -O OpenFlow13 dump-flows s3`. En orden de aparición se tiene la primera entrada que corresponde al flujo de alta prioridad que envía los mensajes LLDP hacia el controlador, la segunda entrada de flujo borra mensajes IPv6 y la última es la entrada de flujo *table-miss*.

```
root@jofnar-HP:/home/jofnar# ovs-ofctl -O OpenFlow13 dump-flows s3
 cookie=0x0, duration=158.567s, table=0, n_packets=529, n_bytes=31740, priority=65535,dl_dst=01:80:c2:00:00:0e
,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=152.075s, table=0, n_packets=5, n_bytes=620, priority=1,ipv6 actions=drop
 cookie=0x0, duration=158.573s, table=0, n_packets=2, n_bytes=184, priority=0 actions=CONTROLLER:65535
```

Figura A.58: Entradas de flujo iniciales en s3

4. El siguiente paso consiste en verificar la correcta conectividad entre los *hosts* de la red. Por tal motivo, se debe abrir una terminal `xterm` de `h1` y ejecutar `ping` hacia la dirección de `h2`, no es necesario `ping` de manera continua. También se puede realizarlo desde la CLI de Mininet mediante el comando `h1 ping -c 1 h2`. Si `ping` establece la comunicación tal como muestra la Figura A.59, de inmediato se procede al análisis del descubrimiento de rutas.
5. **Análisis del descubrimiento de rutas**: inmediatamente después de la prueba de conectividad se debe observar los mensajes en el controlador. En la Figura A.59 se visualiza el ejemplo de salida esperada. Se puede notar que la aplicación encuentra y presenta los caminos disponibles desde `s4` a `s1` y luego de `s1` a `s4`. Cada camino se muestra como un arreglo de números que corresponden a los identificadores de los *switches* en la ruta; seguido se indica el coste, teniendo un valor bajo la ruta más corta y un valor alto para la ruta más larga. Así mismo se muestra el tiempo necesario para instalar las rutas, en el orden de los segundos.
   Las rutas descubiertas por el controlador corresponden a la topología de la Figura A.55. De manera adicional el estudiante deberá volver a consultar las tablas de flujos y evidenciar las entradas de flujo instaladas sobre los *switches*. Por ejemplo, en la Figura A.60 se puede ver las entradas de flujo para `s3` donde se debe ahondar el análisis de las entradas de flujo de tabla de grupo. Existen 4 nuevas entradas, donde 2 de ellas son entradas de flujo de tabla de grupo, que en prácticas previas ha sido analizada su implicación y funcionamiento.

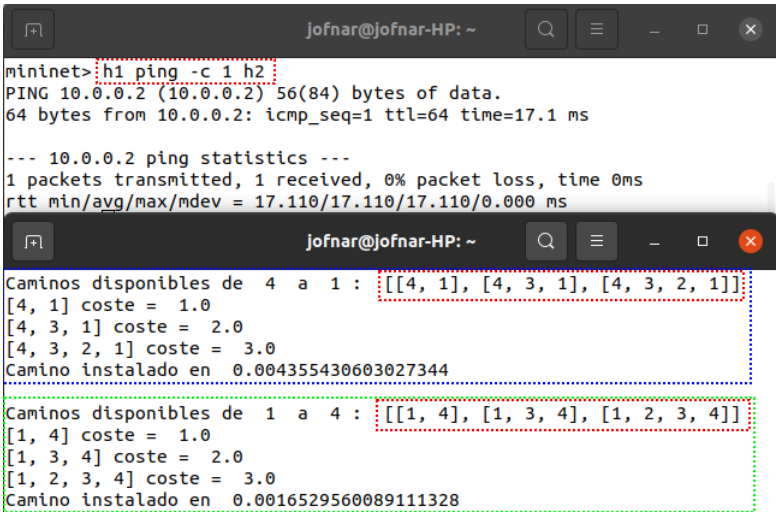


Figura A.59: Mensajes en el Controlador: Rutas descubiertas, coste asociado y tiempo para instalar flujos

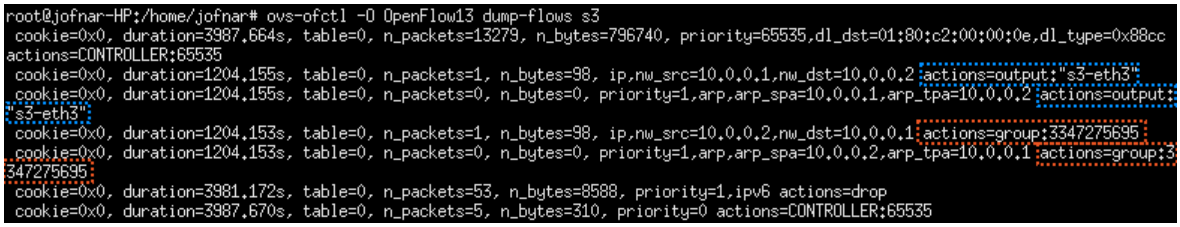


Figura A.60: Nuevas entradas de flujo instaladas en s3.

A.7.6. Preguntas y resoluciones:

Una vez realizada la práctica se puede responder las siguientes interrogantes:

- ¿Qué funciones contiene el código de la aplicación? ¿Qué papel desempeña cada una?
- ¿Existe un evento o función que permita detectar la caída de un enlace o de un *switch*?

A.7.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.8.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.8. Práctica 8: Reenrutamiento rápido

A.8.1. Objetivos:

- Modificar la aplicación de la Práctica 7 para los fines de reenrutamiento rápido.
- Programar registros que se visualicen en el controlador para la administración y monitoreo del estado de los enlaces.
- Instalar entradas de flujo que sirvan como redundancia de tolerancia a fallos ante la caída de un enlace.

A.8.2. Introducción:

En esta práctica se busca modificar una aplicación existente y analizada en una práctica anterior con el fin de implementar la funcionalidad de reenrutamiento rápido y luego evaluar su comportamiento para un escenario de emulación. La topología considerada para esta práctica se presenta en la Figura A.61, que había sido considerada en la Práctica 7, donde la misma aplicación en su funcionalidad de descubrimiento de rutas determinó 3 caminos posibles desde `h1` a `h2`. Cabe recordar que la aplicación en mención es `multicamino.py` y está documentada en el trabajo "*Estudio de técnicas de Ingeniería de Tráfico basadas en SDN*" [1], que en adelante se denominará como "*documentación base*".

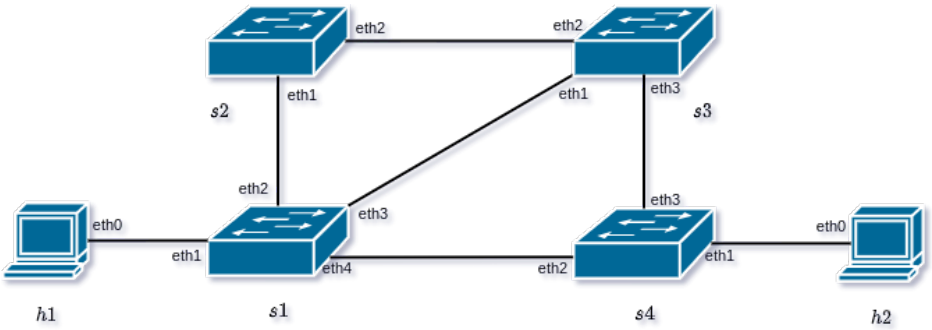


Figura A.61: Topología multicamino propuesta para reenrutamiento rápido.

El Reenrutamiento rápido es una funcionalidad tecnológica soportada por la red tradicional por medio de los protocolos IP y MPLS. El objetivo es proporcionar una inmediata recuperación de la conectividad ante la pérdida del enlace principal o crítico por donde circula el tráfico. Las rutas de respaldo que se configuran para reenrutamiento rápido tienen la misión de:

- Protección de enlaces.
- Protección de nodos.

No cualquier enrutador permite el etiquetado MPLS, por tanto no todos los enrutadores pueden facilitar el reenrutamiento rápido en la red tradicional. Por otra parte, desde las SDN como una red programable, se vuelve más viable la implementación de esta funcionalidad. Esta práctica aborda este desafío, partiendo de una aplicación de descubrimiento de rutas y empleando eventos del controlador en base al Protocolo OpenFlow.

[1] Study of SDN Traffic Engineering techniques: `http://repositorio.unican.es:8080/xmlui/bitstream/handle/10902/14193/409476.pdf`

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[2].
- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[3]
- Controlador RYU.
- Mininet

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.8, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.8: Software utilizado para el desarrollo de la guía Práctica 8

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *switch*	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

A.8.3. Marco teórico a desarrollar:

- Reenrutamiento rápido en la red tradicional IP/MPLS
- Eventos de Ryu para OpenFlow:
  - `event.EventSwitchEnter`
  - `event.EventSwitchLeave`
  - `event.EventLinkAdd`
  - `event.EventLinkDelete`

A.8.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el procedimiento que se describe a continuación, así como, es de vital importancia leer previamente y apoyarse simultáneamente de la documentación base[1] presentada en el apartado anterior.

1. **Crear la topología:** La topología propuesta a implementar en Mininet se muestra en la Figura A.61. Dado que esta topología ya fue implementada en la Práctica 7, se puede utilizarla sin mayor inconveniente, caso contrario debe desarrollar un *script* en Python, `topoMulticamino.py`. La topología consta de 4 *switches* y 2 *hosts*.

[2]Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop
[3]Guía de instalación de Spyder 3 en Linux: https://docs.spyder-ide.org/current/installation.html#linux

2. **Escribir la aplicación para el controlador Ryu:** Para el desarrollo de la aplicación en lenguaje Python, denominada `reenrutamiento.py`, se empleará la aplicación `multicamino.py` desarrollada en la Práctica 7 o también anexa en la documentación base[1]. La aplicación `multicamino.py` debe ser considerablemente modificada para los fines de reenrutamiento rápido, por lo que se debe tomar en cuenta las siguientes instrucciones:

   a) Revisar el código de la aplicación `multicamino.py` empleada en la Práctica 7. Esta aplicación, entre una de sus funcionalidades permite el descubrimiento de rutas, lo que será de utilidad para instalar la ruta principal de tráfico y las rutas de respaldo. Se debe prestar atención a las funciones que permiten el descubrimiento de caminos y otras que refieren a los eventos `event.EventSwitchEnter`, `event.EventSwitchLeave`, `event.EventLinkAdd` y `event.EventLinkDelete`.

   b) Clonar (copiar o transcribir) completamente el código de `multicamino.py` en un nuevo *script*, `reenrutamiento.py`.

   c) En la función `get_optimal_paths` donde se devuelve cierta cantidad de caminos de mínimo coste, se debe modificar el condicional interno para que devuelva todos los caminos.

   d) Dentro de la función `install_paths` se debe eliminar todo lo referente a la instalación de entradas de flujo de grupo, empleadas para balanceo de carga. En su lugar, dentro del bucle que recorre los nodos de cada ruta, se deben instalar flujos de prioridad decreciente llamando a la función `add_flow`. Se pretende dar mayor prioridad al flujo de menor coste, se puede lograr mediante un bucle que reduzca la prioridad a medida que aumente el coste del camino. Estos flujos a agregar son específicamente 2, uno por cada campo de coincidencia (`match_ip` y `match_arp`)

   e) En las funciones `switch_leave_handler`, `switch_enter_handler`, `link_delete_handler` y `link_add_handler`, se debe programar salidas de impresión en pantalla mediante `print` con información o detalles relevantes al *switch* o puerto involucrado en el evento que activa a dichas funciones. Por ejemplo, en el caso de pérdida de un enlace, el evento llama a la función `link_delete_handler`, lo que debe imprimir un mensaje como: `Perdido enlace de SW1 - SW2`.

   f) Para el reenrutamiento ante la caída de un enlace, en la función `link_delete_handler` se debe programar el envío de un mensaje de modificación de flujos (`OFPFlowMod`) al *switch* con instrucciones de borrado de flujos (`command=ofp.OFPFC_DELETE`) coincidentes con el puerto involucrado en la pérdida de enlace.

   g) Por último, en la función `link_add_handler`, que se activa tras el evento `EventLinkAdd`, se tiene que agregar líneas de código que preparen todos los parámetros para llamar la función `install_paths(self,src,first_port,dst,last_port,ip_src,ip_dst)`, modificada anteriormente. Cuando se agrega un nuevo enlace se necesita instalar un flujo para la ruta llamando a la función `install_paths`, por lo que se debe conocer de la ruta: el *switch* de origen (`src`), el primer puerto en la ruta (`first_port`), el *switch* de destino (`dst`), el último puerto en la ruta (`last_port`).

   El algoritmo debe asegurar obtener estos parámetros desde las estructuras de datos almacenadas en la aplicación (diccionario `host` y `arp_table`) por lo que previamente se recomienda comprender estas estructuras y los datos que almacenan. Dado que el enlace que se agrega no siempre pertenecerá a una ruta, e implica que en algún caso no se podrá instalar la ruta,

entonces es idóneo desarrollar el algoritmo dentro de una excepción.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá realizar pruebas de conectividad mediante `ping` y evidenciar el comportamiento de la red ante la caída de un enlace así como el retorno a la ruta principal ante la recuperación del enlace.

A.8.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Ejecutar el controlador con la aplicación desarrollada.
- Verificar la conexión entre *hosts* el reenrutamiento rápido.
- Inhabilitar un enlace y analizar el reenrutamiento rápido por otra ruta.
- Habilitar el enlace inactivo y analizar el comportamiento de recuperación del enlace.

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `sudo python topoMulticamino.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.62, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s4-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s3-eth1 s1-eth4:s4-eth2
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s3-eth2
s3 lo:  s3-eth1:s1-eth3 s3-eth2:s2-eth2 s3-eth3:s4-eth3
s4 lo:  s4-eth1:h2-eth0 s4-eth2:s1-eth4 s4-eth3:s3-eth3
c0
```

Figura A.62: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. Antes de ejecutar el controlador, se debe asegurar que los conmutadores empleen el Protocolo OpenFlow versión 1.3. Con este fin, se abre las terminales `xterm` de cada uno de los *switches* y se ejecuta la línea de comando: `ovs-vsctl set Bridge s1 protocols=OpenFlow13`, donde `s1` corresponde al identificador del *switch*.
3. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `ryu-manager --observe-links reenrutamiento.py`. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al controlador (`packet in`), registros de administración y monitoreo de los enlaces o errores que pueda presentar la aplicación. En la Figura A.63 se puede ver una ejecución limpia, sin errores, de la aplicación `reenrutamiento.py`. En el caso de recibir errores, se deben resolver los mismos antes de continuar.

   En la Figura A.63 se muestran los registros generados por las funciones `link_add_handler` y `switch_enter_handler`, estos registros son informativos, de monitoreo y administración de estados de *switches* y enlaces de la red.

```
jofnar@jofnar-HP:~$ ryu-manager --observe-links P8/reenrutamiento.py
loading app P8/reenrutamiento.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app P8/reenrutamiento.py of ProjectController
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
SW 4 :: Msg Respuesta de Caracteristicas (en CONFIG_DISPATCHER)
SW 1 :: Msg Respuesta de Caracteristicas (en CONFIG_DISPATCHER)
SW 3 :: Msg Respuesta de Caracteristicas (en CONFIG_DISPATCHER)
SW 2 :: Msg Respuesta de Caracteristicas (en CONFIG_DISPATCHER)
SW 4 :: Evento de Ingreso de SW, agregado
SW 1 :: Evento de Ingreso de SW, agregado
SW 2 :: Evento de Ingreso de SW, agregado
SW 3 :: Evento de Ingreso de SW, agregado
Agregado enlace de SW1 - SW2
Agregado enlace de SW3 - SW2
Agregado enlace de SW1 - SW3
Agregado enlace de SW2 - SW3
Agregado enlace de SW4 - SW3
Agregado enlace de SW3 - SW1
Agregado enlace de SW2 - SW1
Agregado enlace de SW4 - SW1
Agregado enlace de SW1 - SW4
Agregado enlace de SW3 - SW4
```

Figura A.63: Registros de mensajes en el controlador tras la ejecución de la aplicación reenrutamiento.py

4. El siguiente paso consiste en verificar la correcta conectividad entre los *hosts* de la red. Por tal motivo, se debe abrir una terminal `xterm` de `h1` y ejecutar `ping` hacia la dirección de `h2`, es necesario `ping` de manera continua. También se puede realizarlo desde la CLI de Mininet mediante el comando `h1 ping h2`. Si `ping` establece la comunicación tal como muestra la Figura A.64, de inmediato se procede al análisis de los flujos instalados en los *switches* adyacentes a los *hosts*.

```
jofnar@jofnar-HP: ~
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=19.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.084 ms
jofnar@jofnar-HP: ~
[4, 1] coste =  1.0
[4, 3, 1] coste =  2.0
[4, 3, 2, 1] coste =  3.0
Camino instalado en  0.0038177967071533203
```

Figura A.64: Mensajes en el Controlador: Rutas descubiertas, coste asociado y tiempo para instalar flujos

5. **Análisis de rutas instaladas, principal y de respaldo**: Inmediatamente después de la prueba de conectividad se deben observar las entradas de flujo instaladas, especialmente en los *switches* críticos que son aquellos adyacentes a los *hosts*. En la Figura A.65 se pueden visualizar las entradas de distinta prioridad, evidenciando que la entrada de flujo de mayor prioridad (`priority=32767` y `priority=9999`) para las coincidencias de paquetes IP y ARP tienen acciones de salida hacia la ruta más corta (`actions=output:"s1-eth4"`) mientras que las otras entradas presentan prioridades menores y servirán de respaldo en el caso de falla del enlace principal.
6. Para verificar el reenrutamiento rápido, se debe provocar la caída del enlace crítico entre $s1$ y $s4$,

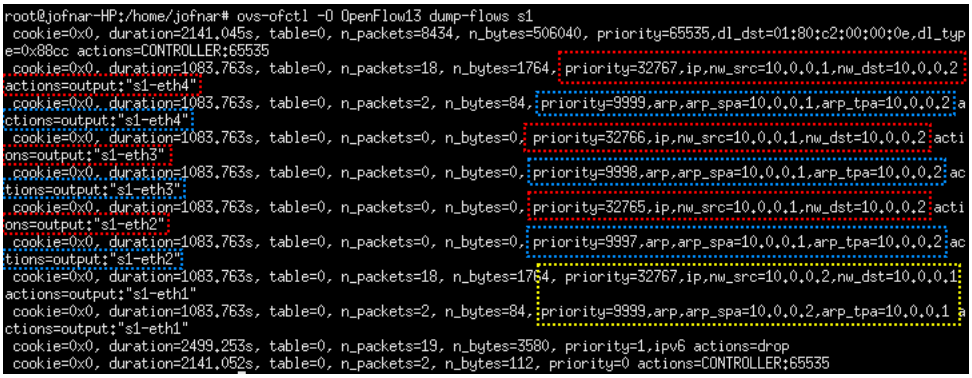


Figura A.65: Entradas de flujo sobre s1 para ruta principal y otras para respaldo

esto se puede hacer desde la CLI de Mininet mediante `link s1 s4 down`. Antes de realizarlo, compruebe que se encuentre en ejecución la prueba de conectividad de ping entre $h1$ y $h2$.

7. **Análisis del reenrutamiento rápido ante la caída del enlace de la ruta principal**: Tras inhabilitar el enlaces `s1-s4`, en la Figura A.66 se puede visualizar los mensajes de administración y monitoreo del estado del enlace y las acciones que se ejecutan. Tal como se programó en la función `link_delete_handler`, se borran las entradas de flujo sobre los *switches* con respecto a los puertos involucrados. El reenrutamiento es transparente al usuario, es así que la prueba de `ping` no se detiene en ningún momento, solo se registra una ligera variación en el tiempo promedio de respuesta.

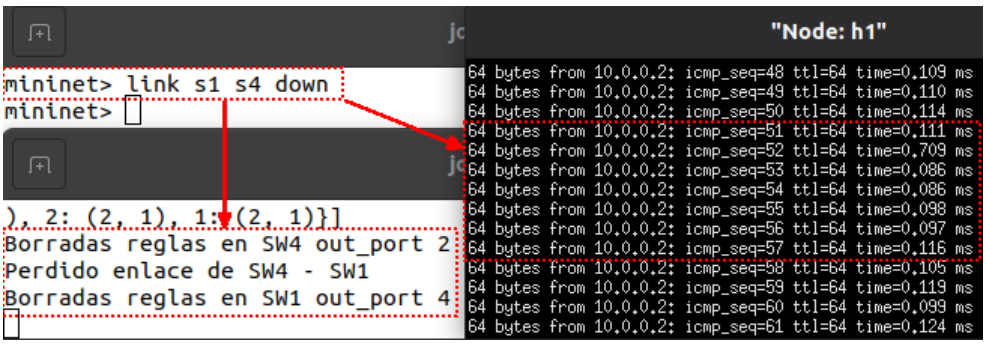


Figura A.66: Comportamiento de la red ante la caída de enlace.

Si se revisan las tablas de entradas flujo sobre `s1`, el estudiante podrá constatar que se han borrado las entradas de flujo de la ruta principal, se podrá comparar con aquellas mostradas en la Figura A.65.

8. Para el siguiente, se busca verificar el reenrutamiento rápido ante la recuperación del enlace, para ello, se debe provocar la reposición del enlace entre $s1$ y $s4$ que fue inhabilitado, esto se puede hacer desde la CLI de Mininet mediante `link s1 s4 up`. Antes de realizarlo, compruebe que se encuentre en ejecución la prueba de conectividad de ping entre $h1$ y $h2$.
9. **Análisis del reenrutamiento rápido ante la reposición del enlace de la ruta principal**: Tras inhabilitar el enlaces `s1-s4`, en la Figura A.67, nuevamente, se puede visualizar los mensajes de administración y monitoreo del estado del enlace y las acciones que se ejecutan, tal como se programó en la función `link_add_handler`, se reinstalan la entrada de flujo de la ruta principal.

El reenrutamiento es transparente al usuario, lo evidencia así la prueba de `ping` que no se detiene en ningún momento, solo se registra una ligera variación en el tiempo promedio de respuesta.

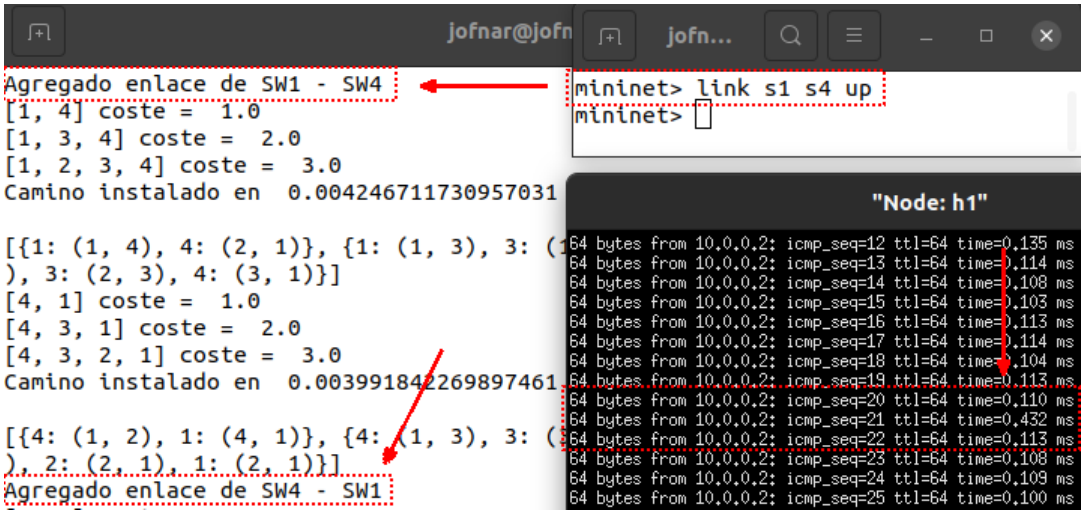


Figura A.67: Comportamiento de la red ante la recuperación del enlace.

Si se revisan nuevamente las tablas de entradas flujo sobre `s1`, el estudiante podrá verificar que se ha agregado las entradas de flujo de la ruta principal, se podrá comparar con aquellas mostradas en la Figura A.66.

A.8.6. Preguntas y resoluciones:

Una vez realizada la práctica se deben responder las siguientes interrogantes:

- ¿Qué tan escalable puede ser el reenrutamiento rápido?
- En la práctica ¿Se ha dado la facultad de reenrutamiento rápido a un solo *switch* o a todos?

A.8.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.8.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.9. Práctica 9: *Firewall* con Ryu

A.9.1. Objetivos:

- Entender en funcionamiento de un Firewall.
- Establecer reglas de seguridad.
- Entender API REST.

A.9.2. Introducción:

En esta práctica se pretende estructurar reglas de seguridad para permitir o bloquear el tráfico entre *hosts*, dichas reglas se implementarán usando la herramienta Postman mediante métodos de API REST. Las reglas se basan en tipo de tráfico y puerto. En el campo práctico el manejo de reglas de seguridad de acceso a un cliente otorga funciones como habilitación o negación de un servicio, que puede ser utilizado para el control de suscripciones. También puede ser usado para filtrar el acceso a contenido específico.

REST es usado para definir APIs. Una API REST también conocida como API RESTful no es un protocolo sino que se trata de restricciones que una API debe cumplir para ser una API REST. Dichas restricciones tienen el objetivo de maximizar la escalabilidad, independencia y interoperabilidad de las interacciones de software, y sobre todo proporciona un medio simple para construir APIs. Los conceptos de acceso basados en web se usan para la interacción entre la aplicación y el servicio [3].

REST se ejecuta sobre el protocolo HTTP que es un protocolo sin estado, es decir, el servidor no conserva registros acerca del estado del cliente lo que proporciona un controlador SDN más eficiente. REST destaca una interfaz uniforme entre los componentes la cual es independiente de la aplicación cliente-servidor implementada en REST. Esto permite que los servicios del controlador evolucionen de manera independiente, al igual que diferentes aplicaciones en distintos idiomas pueden llamar al mismo servicio del controlador a través de una API REST. En REST se presentan distintos tipos de operaciones, estas son: *GET*, *POST*, *PUT*, *DELETE*; cada operación es específica.

- *GET*: Solicita un recurso al servidor sin modificar su estado.
- *POST*: Crea un nuevo recurso en el servidor.
- *PUT*: Actualiza o modifica un recurso existente en el servidor.
- *DELETE*: Elimina recursos en el servidor.

Por otra parte, un *Firewall* proporciona seguridad a la red ya que monitorea el tráfico tanto entrante como saliente y es el encargado de decidir si este pasa o lo bloquea. Para saber qué hacer con el tráfico un *Firewall* se basa en un conjunto de reglas de seguridad. En esta práctica se implementa un *Firewall* de inspección activa que permite o bloquea el tráfico en función del estado, puerto y protocolo. Las decisiones se toman en base a reglas de seguridad las cuales se añaden a los *switches* en forma de entradas de flujo.

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[4].

[4]Página oficial de descarga de Ubuntu: `https://ubuntu.com/download/desktop`

- Conocimientos de nivel intermedio de programación en lenguaje Python, se recomienda el uso del IDE Spyder 3[5]
- Controlador RYU.
- Mininet
- Postman

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.9, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.9: Software utilizado para el desarrollo de la guía Práctica 5

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *Switch*	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

A.9.3. Marco teórico a desarrollar:

- Uso de Postman y sus beneficios.
- Que es una API RESTful y cuales son las ventajas del uso de las mismas.
- Explicar las operaciones de REST y los casos en los que se usan.
- Función de un *Firewall*.

A.9.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

1. **Crear la topología:** La topología de referencia a implementar en Mininet se muestra en la Figura A.68. Para esto se debe desarrollar un *script* en Python, `topologia_firewall.py`. La topología consta de al menos 3 *switches* OpenFlow, 3 *hosts* y al menos dos rutas alternativas.
2. **Establecimiento de reglas de seguridad mediante API REST:** Para el desarrollo de esta práctica se hará uso de la aplicación proporcionada por Ryu `rest_firewall` y el estudiante deberá crear reglas de seguridad usando Postman, se debe tomar en cuenta las siguientes instrucciones:
   a) Revisar la aplicación `rest_firewall` presentada y explicada en el Libro RYU SDN Framework[6].
   b) Habilitar el firewall en cada uno de los *switches* de tal manera que se inhabiliten todas las comunicaciones.
   c) Crear una regla de seguridad que permita realizar ping entre `h1 y h2` y viceversa.
   d) Permitir el paso de todo tipo de tráfico entre `h1 y h3` y viceversa.

[5]Guía de instalación de Spyder 3 en Linux: `https://docs.spyder-ide.org/current/installation.html#linux`
[6]RyuBook: `https://osrg.github.io/ryu-book/en/Ryubook.pdf`

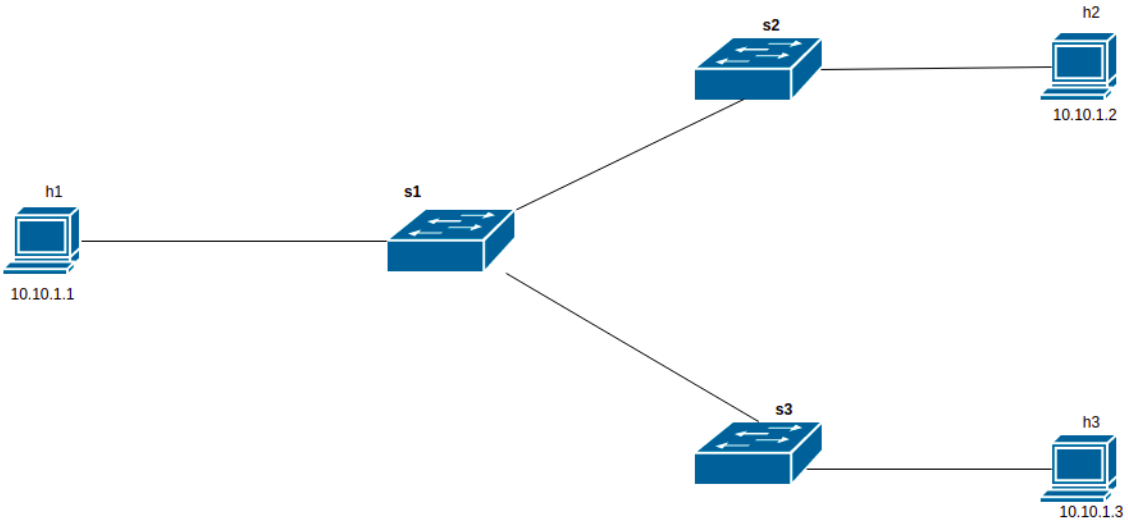


Figura A.68: Topología para direccionamiento de tráfico.

e) Bloquear el tráfico TCP puerto 80 (HTTP) hacia `h3`.
f) Cada una de las reglas antes mencionadas deben ser ejecutadas desde Postman.
g) Indicar las reglas creadas en cada uno de los *switches*.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá generar tráfico de prueba en los *hosts* h1 y h2; y verificar si se cumplen las reglas de seguridad establecidas.

A.9.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Ejecutar el controlador con la aplicación `rest-firewall`.
- Habilitar el *Firewall* en cada uno de los *switches*.
- Verificar que se hayan cortado todas las comunicaciones.
- Establecimiento de la regla 1 y verificación de su funcionamiento.
- Establecimiento de la regla 2 y verificación de su funcionamiento.
- Establecimiento de la regla 3 y verificación de su funcionamiento.

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `$sudo python topologia-firewall.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.69, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados.

```
mininet> net
h1 h1-eth1:s1-eth3
h2 h2-eth1:s2-eth2
h3 h3-eth1:s3-eth2
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s3-eth1 s1-eth3:h1-eth1
s2 lo:  s2-eth1:s1-eth1 s2-eth2:h2-eth1
s3 lo:  s3-eth1:s1-eth2 s3-eth2:h3-eth1
c1
```

Figura A.69: Salida del comando *net* en Mininet posterior a ejecutar la topología

2. En una segunda terminal se ejecuta la aplicación con el controlador Ryu mediante el comando: `$ryu-manager ryu.app.rest_firewall.py`. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al controlador o errores que pueda presentar la aplicación. En la Figura A.70 se puede ver una ejecución sin errores, de la aplicación `rest_firewall.py`, al igual que se observan los dispositivos que se unen como *Firewall*. En el caso de recibir errores, se deben resolver los mismos antes de continuar.

```
jhoss@jhossPC:~$ ryu-manager ryu.app.rest_firewall
loading app ryu.app.rest_firewall
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.rest_firewall of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(9689) wsgi starting up on http://0.0.0.0:8080
[FW][INFO] dpid=0000000000000001: Join as firewall.
[FW][INFO] dpid=0000000000000002: Join as firewall.
[FW][INFO] dpid=0000000000000003: Join as firewall.
```

Figura A.70: Ejecución de la aplicación *rest_firewall* con el controlador RYU.

3. El siguiente paso consiste en habilitar el *Firewall* en cada uno de los *switches* de tal manera que se corten todo tipo de comunicaciones. Posterior a la habilitación, se puede realizar la respectiva verificación mediante una operación *GET*. La salida esperada se indica en la Figura A.71. También se verifica que no hay conectividad entre los *hosts* de la red y los registros en el controlador indican los paquetes bloqueados, esto se evidencia en la Figura A.81.

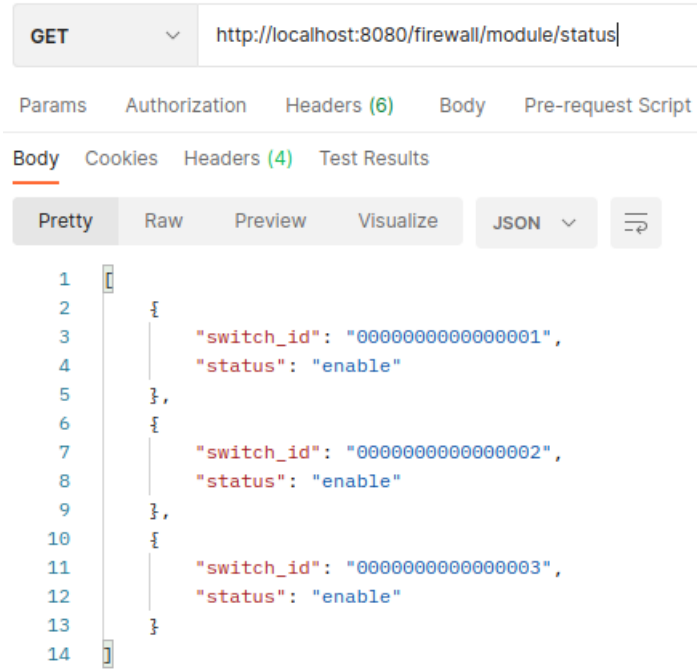


Figura A.71: Estado de *Firewall* en los *switches*

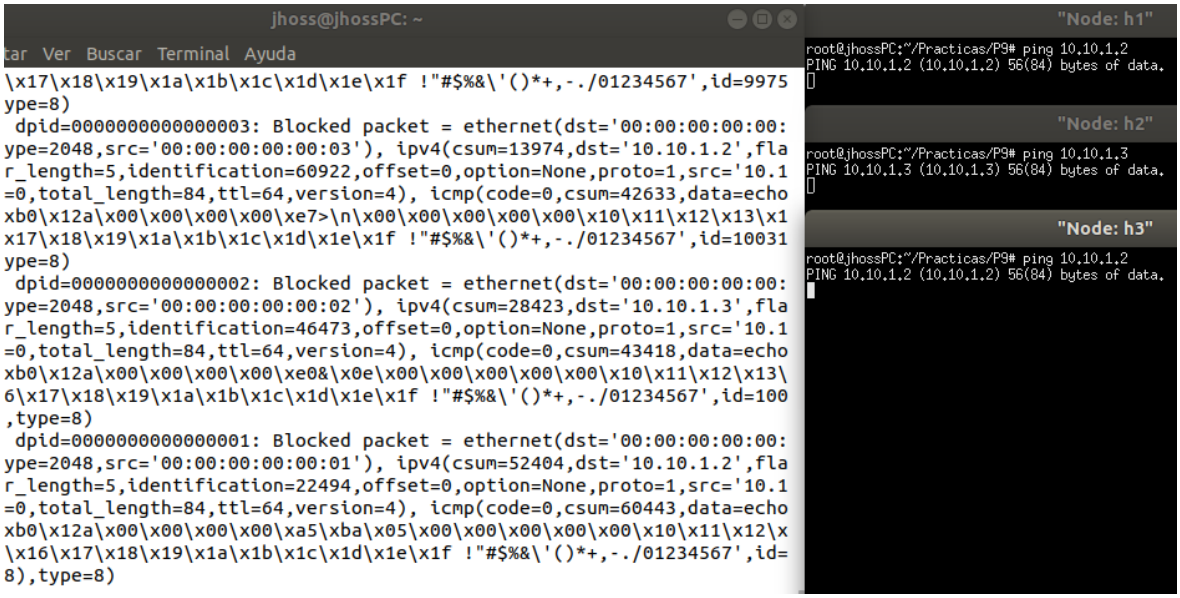


Figura A.72: Bloqueo de toda comunicación

4. **Establecimiento de la primera regla de seguridad y resultado de la misma.**: Mediante Postman y la operación *POST* se deben crear las reglas de seguridad en `s1 y s2`, cabe indicar de que las reglas se crearan en los *switches* como entradas de flujo; posterior a crear y ejecutar la primera regla, ya se permite tráfico ICMP entre h1 y h2, esto se evidencia en la Figura A.73.

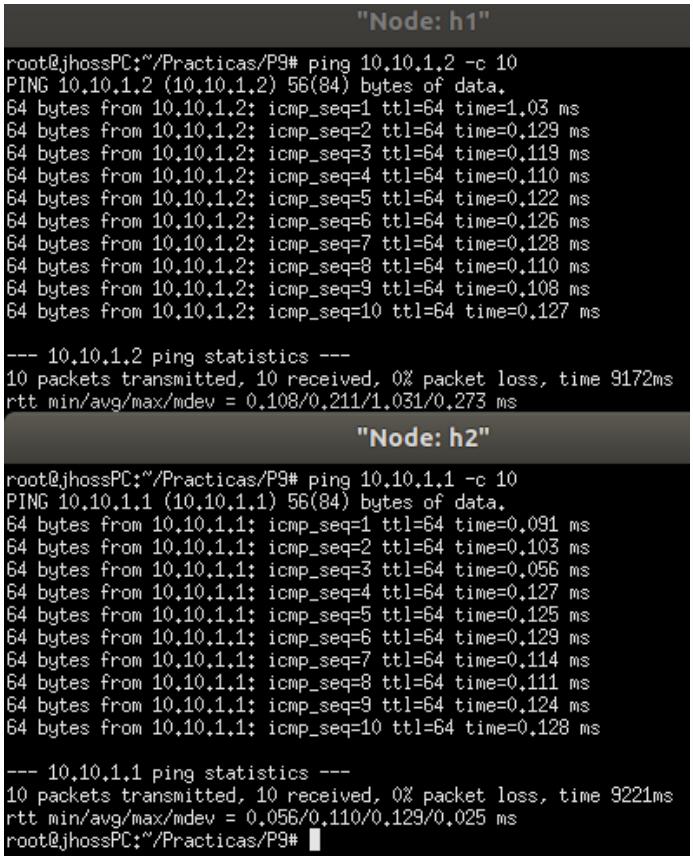


Figura A.73: Resultado de la ejecución de la primera regla de seguridad

5. **Establecimiento de la segunda regla de seguridad y resultado de la misma.**: Mediante Postman y la operación *POST* se deben crear las reglas de seguridad en `s1 y s3`, dicha regla debe permitir el curso de todo tipo de tráfico entre `h1 y h3`, para verificar el establecimiento de la regla se procede a realizar pruebas de conectividad mediante ping entre `h1 y h3`, también realizan pruebas de tráfico TCP mediante iPerf; el resultado esperado se evidencia en las Figuras A.74 y A.75.

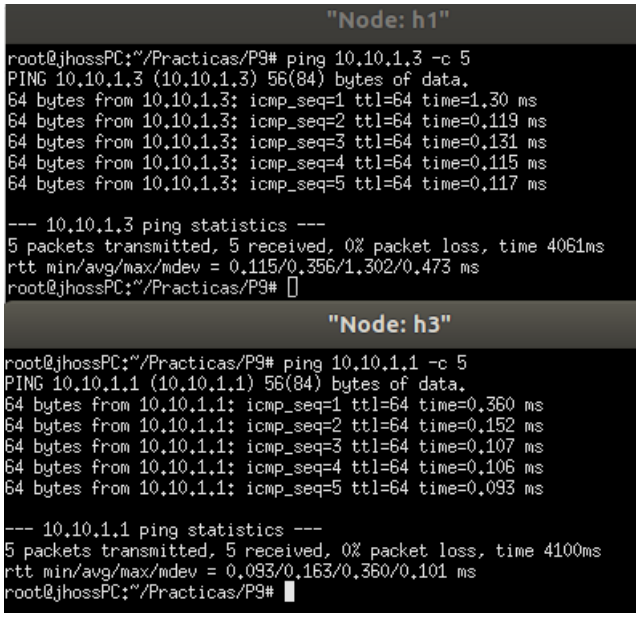


Figura A.74: Resultado de la ejecución de la segunda regla de seguridad (Tráfico ICMP)

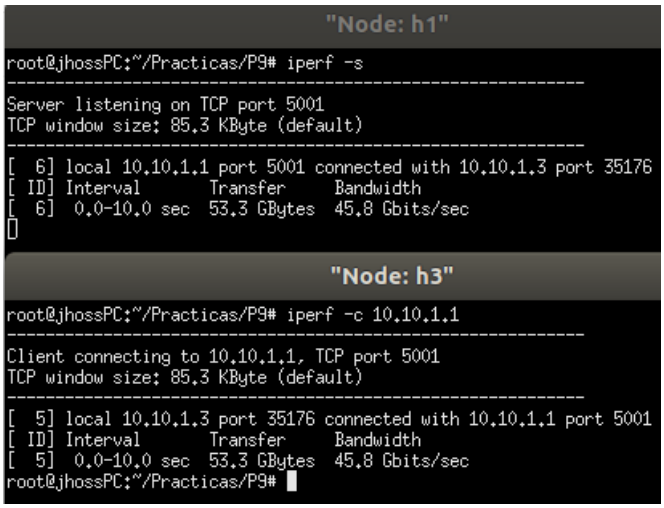


Figura A.75: Resultado de la ejecución de la segunda regla de seguridad (Tráfico TCP)

6. **Establecimiento de la tercera regla de seguridad y resultado de la misma.**: Mediante la operación *POST* se crea las regla de seguridad en `s3`, dicha regla bloquea el tráfico HTTP (TCP puerto 80) tanto entrante como saliente en `s3`, para verificar el establecimiento de la regla se realizan pruebas de tráfico HTTP mediante iPerf; el resultado esperado se evidencia en la Figura A.76, en donde se observa que no circula el tráfico HTTP y en el lado del controlador se observan los paquetes bloqueados.

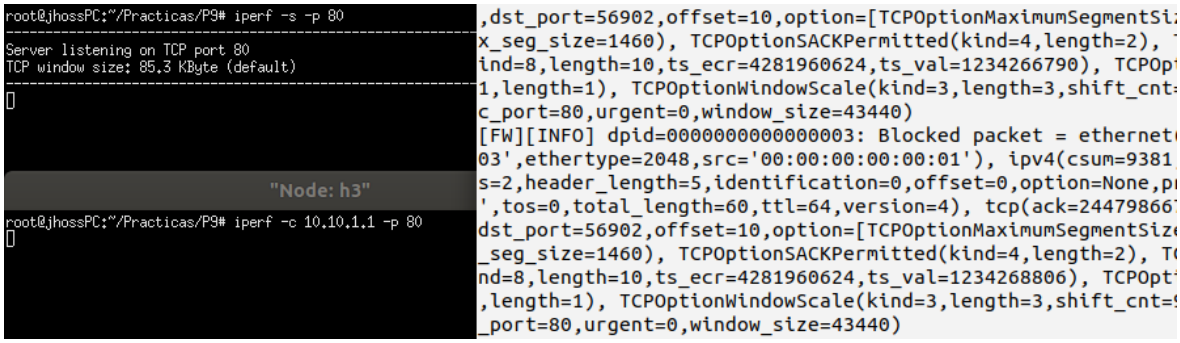


Figura A.76: Resultado de la ejecución de la tercera regla de seguridad (Tráfico HTTP)

A.9.6. Preguntas y resoluciones:

Una vez realizada la práctica se pueden responder las siguientes interrogantes:

- ¿Qué otros criterios se podrían usar para definir las reglas?
- Sin considerar los antes mencionados, indicar casos prácticos en donde se pueda implementar un *Firewall*.

A.9.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.5.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.10. Práctica 10: Seguridad con ODL, *Firewall* y AAA

A.10.1. Objetivos:

- Incursionar en el funcionamiento y manejo del controlador ODL.
- Entender el funcionamiento básico de NIC.
- Crear un *Firewall* simple utilizando NIC.
- Realizar operaciones básicas usando AAA (creación de usuario)

A.10.2. Introducción:

En esta práctica se busca que el estudiante se relacione con el controlador OpenDayligth (ODL), para lo cual se realizan acciones usando el plugin de AAA (Authentication, Authorization and Accounting) y NIC (Network Intent Composition), el estudiante deberá implementar un firewall simple (bloqueo de tráfico) mediante intentos; también deberá crear un nuevo usuario y realizar el proceso necesario para que éste tenga acceso al controlador ODL como administrador. En aplicaciones prácticas tanto NIC como AAA contribuyen en aspectos de seguridad. Mediante AAA se puede definir que usuario tiene acceso al controlador y cómo accederá. Es decir, el usuario podría ingresar al controlador pero no cuenta con el permiso para realizar alguna modificación que altere a la red, mientras que otro usuario (administrador) puede acceder y realizar la modificaciones que crea pertinentes.

NIC es una interfaz de intención que permite a los clientes expresar un estado deseado, una intención es ampliada en base a reglas, políticas y configuraciones de implementación local. NIC es un modelo de intención que transforma el estado deseado en los recursos bajo el control de OpenDayligth. Al encargado de pasar una intención a una implementación se le conoce como renderizador. Para establecer intentos se tienen dos tipos de acciones que son: *ALLOW* y *BLOCK*. *ALLOW* indica que tráfico puede cursar entre los puntos finales, mientras que *BLOCK* impide dicho tráfico. El usuario es capaz de interactuar con NIC ya sea mediante la interfaz RESTful mediante API REST o mediante la CLI de la consola karaf.

AAA ayuda a mejorar la seguridad de OpenDaylight, en la versión de OpenDayligth (Boron) está disponible la autenticación, pues la autorización es experimental y la contabilidad es un trabajo en progreso. La mayoría de aplicaciones están protegidas mediante AAA. AAA permite el control de acceso a los recursos, hace cumplir las políticas para usar dichos recursos y audita su uso. La autenticación proporciona una manera de identificar un usuario. En base a un usuario válido a una contraseña, AAA compara credenciales de autenticación de un usuario con otras credenciales de usuario registradas en una base de datos. Existen diferentes términos y definiciones que se abordan en AAA entre ellos:

- Token: demanda de acceso a un grupo de recursos en el controlador.
- Dominio: grupo de recursos, directos o indirectos, físicos, lógicos, o virtuales que tienen el propósito de controlar el acceso.
- Usuario: persona que posee acceso a un recurso o grupo de recursos en el controlador.
- Rol: única representación de un conjunto de permisos ya sea como administrador o invitado.
- Credencial: prueba de identidad como usuario, contraseña, datos biométricos, entre otros.
- Cliente: un servicio o aplicación que requiere acceso al controlador.
- Grant: entidad que asocia a un usuario con su rol y dominio.
- Claim: conjunto de datos validados con respecto a un usuario.

AAA es habilitado mediante la instalación de `odl-aaa-shiro`.
Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[1].
- Controlador ODL (versión Beryllium SR4) [2].
- Mininet

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.10, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.10: Software utilizado para el desarrollo de la guía Práctica 10

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *switch*	Open vSwitch	2.9.8
Controlador SDN	ODL	Berrylium SR4
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6

A.10.3. Marco teórico a desarrollar:

- Controlador ODL.
- NIC y sus funcionalidades.
- AAA y sus funcionalidades.

A.10.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

1. **AAA(usando el *script* etc/idmtool):**
   - Revisar la guía disponible en [3], pues contiene material de ayuda en el desarrollo de la práctica.
   - Agregar un usuario.
   - Listar los usuarios existentes.
   - Verificar si el usuario está autorizado para acceder al controlador.
   - Observar los roles existentes.
   - Añadir un grant al usuario creado de tal manera que este le permita acceder al controlador.
2. **NIC:**

[1] Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop
[2] Página oficial de descarga de OpenDayligth: https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/
[3] Guía Oficial AAA OpenDayligth: https://docs.opendaylight.org/en/stable-nitrogen/user-guide/authentication-and-authorization-services.html

- Revisar la guía disponible en [4], pues contiene material de ayuda en el desarrollo de la práctica.
- **Crear la topología:** La topología de referencia a implementar en Mininet se muestra en la Figura A.77. Para esto se debe desarrollar un *script* en Python, `topologia_NIC.py`. La topología consta de al menos 3 *switches* OpenFlow, 3 *hosts* y al menos dos rutas alternativas.
- Instalar las funciones: `odl-nic-core-mdsal, odl-nic-console, odl-nic-renderer-of` en el controlador.
- Instalar los intentos entre `h1-h2` y `h1-h3` para bloquear la comunicación entre los mismos.

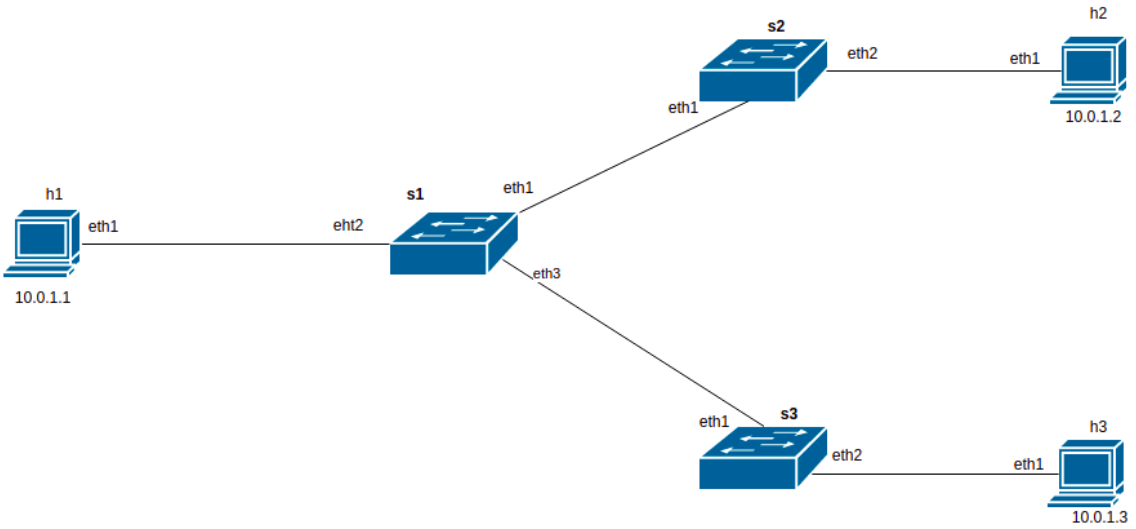


Figura A.77: Topología para *Firewall* mediante NIC.

3. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá generar tráfico de prueba en los *hosts* h1 y h2; al igual que ingresar al controlador ODL.

A.10.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- **AAA:**
- Listar los usuarios existentes.
- Verificar si el usuario está autorizado para acceder al controlador.
- Acceder al controlador.
- **NIC:**
- Ejecutar la topología creada.
- Verificar el bloqueo de comunicación mediante intentos.
- Eliminar intentos y restablecer la comunicación.

[4]Guía Oficial NIC OpenDayligth: https://test-odl-docs.readthedocs.io/en/latest/developer-guide/network-intent-composition-(nic)-developer-guide.html

1. En la dirección en la que se encuentra la carpeta correspondiente al controlador, utilizar el *script* `etc/idmtool` para manipular el IdM TokenAuthRealm. Crear un nuevo usuario en este caso se creó el usuario `usrtesis`, este usuario debe constar en la lista de usuarios, esto se evidencia en la Figura A.78.


```
list_users

command succeeded!

json:
{
    "users": [
        {
            "description": "admin user",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "admin",
            "password": "**********",
            "salt": "**********",
            "userid": "admin@sdn"
        },
        {
            "description": "user user",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "user",
            "password": "**********",
            "salt": "**********",
            "userid": "user@sdn"
        },
        {
            "description": "",
            "domainid": "sdn",
            "email": "",
            "enabled": true,
            "name": "usrtesis",
            "password": "**********",
            "salt": "**********",
            "userid": "usrtesis@sdn"
        }
    ]
}
```


Figura A.78: Lista de usuarios

2. Al crear el usuario este no posee ningún permiso (grant) asignado por lo que no tendrá acceso al controlador, esto se evidencia en la Figura A.79, por lo que al ingresar a `http://localhost:8181/index.html` con el usuario creado no se tendrá acceso alguno. Para que el usuario creado tenga acceso al controlador se debe observar los permisos existentes y asignarle el acceso como administrador. Realizado esto verificar que ya se tenga acceso al controlador como se indica en la Figura A.80.

```
Enter host password for user 'usrtesis/usrtesis':
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8181 (#0)
* Server auth using Basic with user 'usrtesis/usrtesis'
> GET /restconf/streams/ HTTP/1.1
> Host: localhost:8181
> Authorization: Basic dXNydGVzaXMvdXNydGVzaXM6dXNydGVzaXM=
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 401 Unauthorized
< Set-Cookie: rememberMe=deleteMe; Path=/restconf; Max-Age=0; Expires=Wed, 11-Aug-2021 18:39:
26 GMT
* Authentication problem. Ignoring this.
< WWW-Authenticate: BASIC realm="application"
< Content-Length: 0
< Server: Jetty(8.1.15.v20140411)
<
* Connection #0 to host localhost left intact
```

Figura A.79: Usuario sin acceso al controlador

Figura A.80: Ingreso al controlador OpenDayligth desde el usuario creado (usrtesis).

3. A continuación se detallan algunos de los resultados esperados para la implementación de un *Firewall* simple usando NIC. Inicialmente se deben crear intentos que bloqueen la comunicación bidireccional entre `h1-h2` y `h1-h3`, creados los intentos se realizan pruebas de conectividad tanto entre `h1-h2` (marcado de verde) y `h1-h3` (marcado de azul), esto se videncia en la Figura A.81.

```
mininet> h1 ping h2 -c 5
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.

--- 10.0.1.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms

mininet> h1 ping h3 -c 5
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.

--- 10.0.1.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms
```

Figura A.81: Comunicación bloqueada mediante intentos

4. Posteriormente, se debe realizar la eliminación de los intentos que bloquean la comunicación entre `h1-h3`, de tal manera que la comunicación entre dichos *hosts* se restablezca, esto se evidencia en la Figura A.82 (marcado de azul) y se puede verificar que la comunicación entre `h1-h2` se mantiene bloqueada.

```
mininet> h1 ping h2 -c 5
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.

--- 10.0.1.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4092ms

mininet> h1 ping h3 -c 5
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=1 ttl=64 time=0.827 ms
64 bytes from 10.0.1.3: icmp_seq=2 ttl=64 time=0.135 ms
64 bytes from 10.0.1.3: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.0.1.3: icmp_seq=4 ttl=64 time=0.134 ms
64 bytes from 10.0.1.3: icmp_seq=5 ttl=64 time=0.133 ms

--- 10.0.1.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4055ms
rtt min/avg/max/mdev = 0.060/0.257/0.827/0.286 ms
```

Figura A.82: Comunicación restablecida entre *h1-h3*

A.10.6. Preguntas y resoluciones:

Una vez realizada la práctica se deben responder las siguientes interrogantes:

- Desde esta experiencia, ¿Cuál es la diferencia entre un *Firewall* implementado en RYU y el implementado en esta práctica?
- En base a lo desarrollado, ¿Qué permiso se debería asignar a un usuario para que este tenga acceso al controlador pero no pueda realizar alguna modificación dentro del mismo?

A.10.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.10.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.11. Práctica 11: Compatibilidad de BGP ODL con enrutadores BGP tradicionales

A.11.1. Objetivos:

- Conectar diferentes sistemas autónomos.
- Entender el manejo del plugin BGP de ODL.
- Emplear BGP para interconectar un enrutador BGP ODL a una red tradicional.

A.11.2. Introducción:

En esta práctica se utilizará el plugin BGP de OpenDayligth para interconectar una red SDN con una red tradicional mediante el protocolo BGP. Se tendrá un enrutador (eBGP) el cual trabajará con ODL y se conectaría a una red tradicional compuesta por enrutadores, *hosts* y *switches*, dando origen a una red híbrida. Esta práctica evidencia el primer paso para migrar a futuro, desde una red tradicional hacia una red SDN; está basada en informacion impartida en el curso `SDN Opendayligth(ODL) Controller Crash Course` disponible en [1].
Para la elaboración de esta práctica se usará Containernet que extiende el nivel de emulación con respecto a Mininet pues permite el uso de contenedores Docker como instancias de cómputo dentro de la red emulada, lo que no es posible realizar con Mininet. Dichos Dockers permiten la implementación BIRD Internet Routing Daemon (BIRD).
BIRD es una implementación de código abierto para el enrutamiento de paquetes IP en sistemas similares a Unix, BIRD utiliza protocolos de enrutamiento como BGP, Routing Information Protocol (RIP) y Open Shortest Path First (OSPF); e incluso ha llegado a reemplazar a Quagga. BIRD implementa una tabla de enrutamiento interna que se conecta a los protocolos que se admiten, dichos protocolos importan y exportan rutas de red hacia y desde dicha tabla interna, intercambiando información entre diferentes protocolos de enrutamiento. La tabla interna de BIRD se puede conectar a la tabla de enrutamiento del Kernel, lo que permite exportar rutas de red desde la tabla interna a la tabla de enrutamiento del kernel y también aprender sobre las rutas de red desde la tabla de enrutamiento del kernel e importar estas rutas a su tabla de enrutamiento interna [42].

OpenDayligth contiene el complemento OpenDaylight (BGP) que permite la conexión entre sistemas autónomos (AS). La función principal de BGP es intercambiar rutas entre ASs. Este complemento permite el intercambio de tráfico externo, para intercambiar información de enrutamiento entre dos sistemas BGP, se requiere configurar un emparejamiento entre routers BGP, se establece una conexión TCP entre los dos sistemas y se intercambian mensajes para abrir y confirmar parámetros de conexión e intercambio de rutas. [2]

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[3].

[1] Curso SDN Opendayligth(ODL) Controller Crash Course: `udemy.com`
[2] docs.opendaylight.org/projects/bgpcep/en/stable-fluorine/bgp/bgp-user-guide-bgp-peering.htmlexternal-peering-configuration
[3] Página oficial de descarga de Ubuntu: `https://ubuntu.com/download/desktop`

- Controlador ODL (versión Sodium) [4].
- Containernet [5]

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.11, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

Tabla A.11: Software utilizado para el desarrollo de la guía Práctica 11

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Containernet	estandar
Virtualizador de *Switch*	Open vSwitch	2.9.8
Controlador SDN	ODL	Sodium
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6

A.11.3. Marco teórico a desarrollar:

- Complemento Opendayligth BGP y sus funcionalidades.
- Containernet.
- BIRD
- Instancias BGP.

A.11.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

1. Revisar la documentación sobre cómo utilizar el complemento BGP de OpenDayligth[6]. Esta documentación describe una guía de como utilizar dicho complemento, aportando información de utilidad para el desarrollo de la práctica.
2. Revisar la documentación de configuración de BIRD[7]. Dicha documentación aporta con información relevante para el desarrollo de la práctica.
   Crear archivos de configuración BIRD para los routers (R1, R2, R3): Se deben crear los archivos .conf, estos incluyen la configuración del protocolo BGP en cada sistema autónomo; en dichos archivos se deben indicar los campos:
   - router id.
   - Protocol Kernel.
   - Protocol Device.
   - Configuración BGP
3. **Crear la topología:** La topología de referencia a implementar en Containernet se muestra en la Figura A.83, se debe crear solamente la topología correspondiente a la red tradicional (AS 65100,

[4]Página oficial de descarga de OpenDayligth: https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/opendaylight/
[5]Página oficial de descarga de Containernet: https://github.com/containernet/containernet
[6]https://docs.opendaylight.org/projects/bgpcep/en/stable-fluorine/bgp/bgp-user-guide-rib-config-policies.html
[7]https://bird.network.cz/?get$_docv = 20f = bird - 3.html$

AS 65200 y AS 65400). Para esto se debe desarrollar un *script* en Python, `topologiaBGP.py`. La topología consta de 3 enrutadores, 2 *switches* y 3 *hosts*. Cabe señalar que en la topología se debe tomar en cuenta los archivos de configuración BIRD creados.

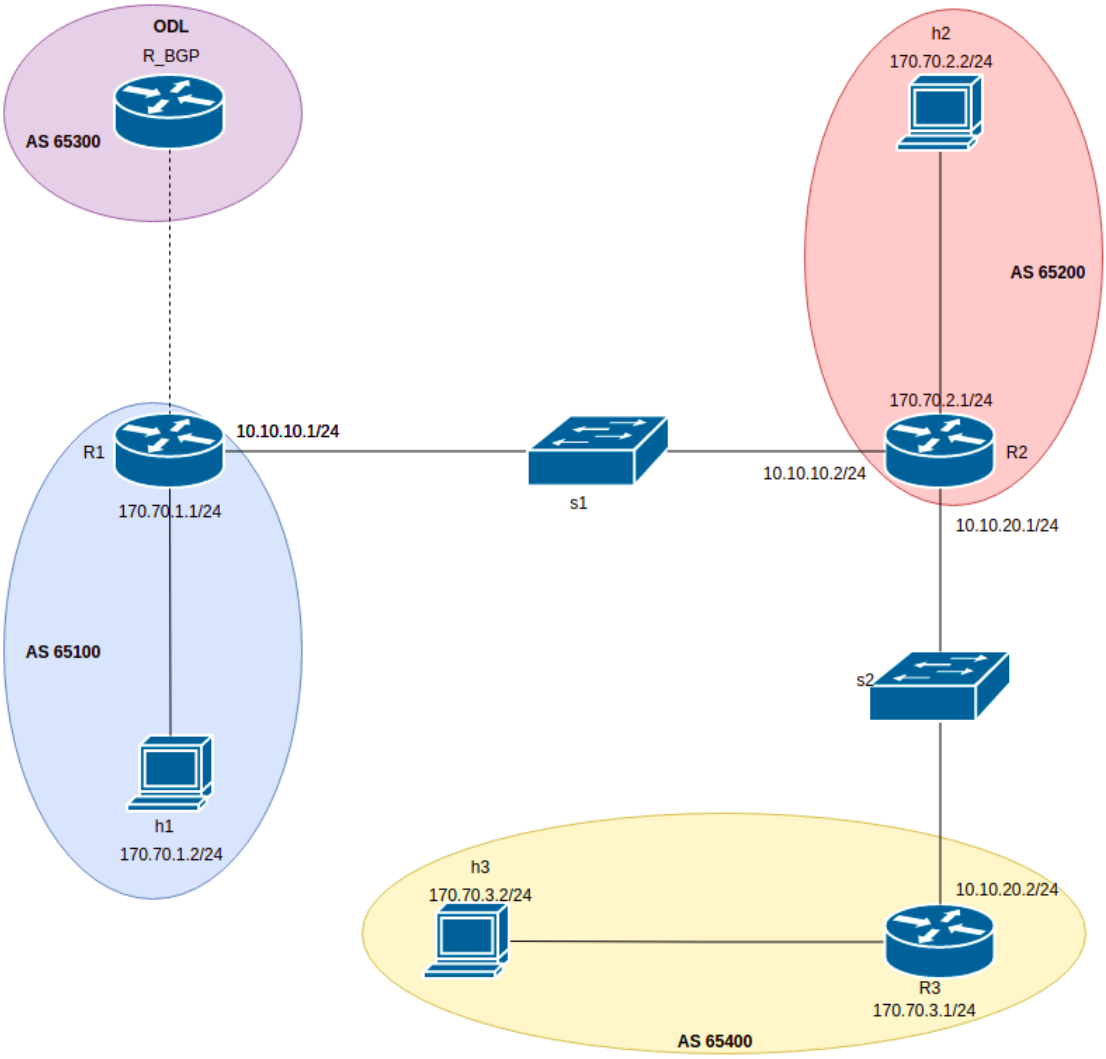


Figura A.83: Topología para BGP.

4. Instalar el plugin BGP mediante el comando `feature:install odl-bgpcep-bgp` en el controlador OpenDayligth.
5. Crear un *script* .xml en donde se cree una instancia BGP para definir el enrutador eBGP y su sistema autónomo, en este *script* se deben especificar parámetros como: `name`, `router-id` y `as`.
6. Crear solicitudes de emparejamiento BGP mediante un *script* .xml en donde se deben establecer los campos `neighbor-address`, `peer-type` y `peer-as`.
7. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberá visualizar detalles de rutas en el enrutador ODL y analizar paquetes en Wireshark.

A.11.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar la topología creada.
- Verificar el estado del protocolo BGP en los enrutadores.
- Crear la instancia ODL BGP mediante API REST.
- Crear la instancia vecino BGP mediante API REST.
- Verificar el estado de protocolo BGP y establecimiento de vecindad.
- Verificar los detalles BGP en el router ODL.

1. En una primera terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `$sudo python topologia_BGP.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.84, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados.

```
containernet> net
r1 r1-eth0:h1-eth0 r1-eth1:s1-eth1
r2 r2-eth0:h2-eth0 r2-eth1:s1-eth2 r2-eth2:s2-eth1
r3 r3-eth0:h3-eth0 r3-eth1:s2-eth2
h1 h1-eth0:r1-eth0
h2 h2-eth0:r2-eth0
h3 h3-eth0:r3-eth0
s1 lo:  s1-eth1:r1-eth1 s1-eth2:r2-eth1
s2 lo:  s2-eth1:r2-eth2 s2-eth2:r3-eth1
```

Figura A.84: Salida del comando *net* en Containernet luego de ejecutar la topología.

2. Verificar el estado del protocolo BGP en los enrutadores `R1, R2, R3`, esto se realiza mediante los comandos `r1 birdc show protocols`, el resultado esperado de esto se evidencia en la Figura A.85, en donde se visualiza que R1 no tiene una conexione establecida con el router ODL BGP.
3. Para que exista conectividad entre R1 y el enrutador ODL perteneciente a la red SDN se deben crear las instancias ODL BGP y vecino BGP mediante API REST, para lo cual se usa la herramienta curl y el método *POST*.
   La instancia ODL BGP, se crea en la dirección: *http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/* y se usa como datos HTTP *POST* el archivo `router_bgp.xml` que es el archivo creado para definir el enrutador BGP ODL.

   La instancia de vecino BGP, se crea en la dirección: *http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocolsprotocols/protocol/openconfig-policy-types:BGP/bgp-odl-router/bgp/neighbors/*, y se usa como datos HTTP *POST* el archivo `vecino_bgp.xml` que es la solicitud de emparejamiento creada.

```
containernet> r1 birdc show protocols
BIRD 1.4.0 ready.
name     proto    table    state  since       info
direct1  Direct   master   up     02:08:55
kernel1  Kernel   master   up     02:08:55
device1  Device   master   up     02:08:55
R2       BGP      master   up     02:08:59    Established
ODl      BGP      master   start  02:08:55    Active        Socket: Connection r
efused
containernet> r2 birdc show protocols
BIRD 1.4.0 ready.
name     proto    table    state  since       info
direct1  Direct   master   up     02:08:55
kernel1  Kernel   master   up     02:08:55
device1  Device   master   up     02:08:55
R1       BGP      master   up     02:08:59    Established
R3       BGP      master   up     02:08:59    Established
containernet> r3 birdc show protocols
BIRD 1.4.0 ready.
name     proto    table    state  since       info
direct1  Direct   master   up     02:08:56
kernel1  Kernel   master   up     02:08:56
device1  Device   master   up     02:08:56
R2       BGP      master   up     02:09:00    Established
```

Figura A.85: Verificación del estado del protocolo BGP en los enrutadores

4. Posterior a crear las instancias se vuelve a verificar el estado del protocolo BGP en `R1,R2,R3`, esto se evidencia en la Figura A.86, en donde se ve que la conexión entre R1 y el enrutador ODL BGP se ha establecido.

```
containernet> r1 birdc show protocols
BIRD 1.4.0 ready.
name     proto    table    state  since       info
direct1  Direct   master   up     02:08:56
kernel1  Kernel   master   up     02:08:56
device1  Device   master   up     02:08:56
R2       BGP      master   up     02:09:00    Established
ODl      BGP      master   up     02:37:52    Established
containernet> r2 birdc show protocols
BIRD 1.4.0 ready.
name     proto    table    state  since       info
direct1  Direct   master   up     02:08:55
kernel1  Kernel   master   up     02:08:55
device1  Device   master   up     02:08:55
R1       BGP      master   up     02:08:59    Established
R3       BGP      master   up     02:08:59    Established
containernet> r3 birdc show protocols
BIRD 1.4.0 ready.
name     proto    table    state  since       info
direct1  Direct   master   up     02:08:56
kernel1  Kernel   master   up     02:08:56
device1  Device   master   up     02:08:56
R2       BGP      master   up     02:09:00    Established
```

Figura A.86: Verificación de protocolo BGP después de haber establecido las instancias BGP

5. En este paso se deben verificar los detalles BGP en el enrutador ODL, para esto se ejecuta el comando: *curl -v –user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X GET http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/rib/bgp-odl-router/ | xmllint –format - >resultado.xml*, en donde la salida de esta consulta se guardará en un archivo `resultado.xml`, que se evidencia en la Figura A.87. Aquí se observa que se ha aprendido las direcciones de los sistemas autónomos correspondientes a la red tradicional.

Figura A.87: Verificación de detalles BGP en el router ODL

6. Mediante Wireshark se realiza la captura de paquetes de tal manera que se puedan visualizar los paquetes BGP, como se evidencia en la Figura A.88.

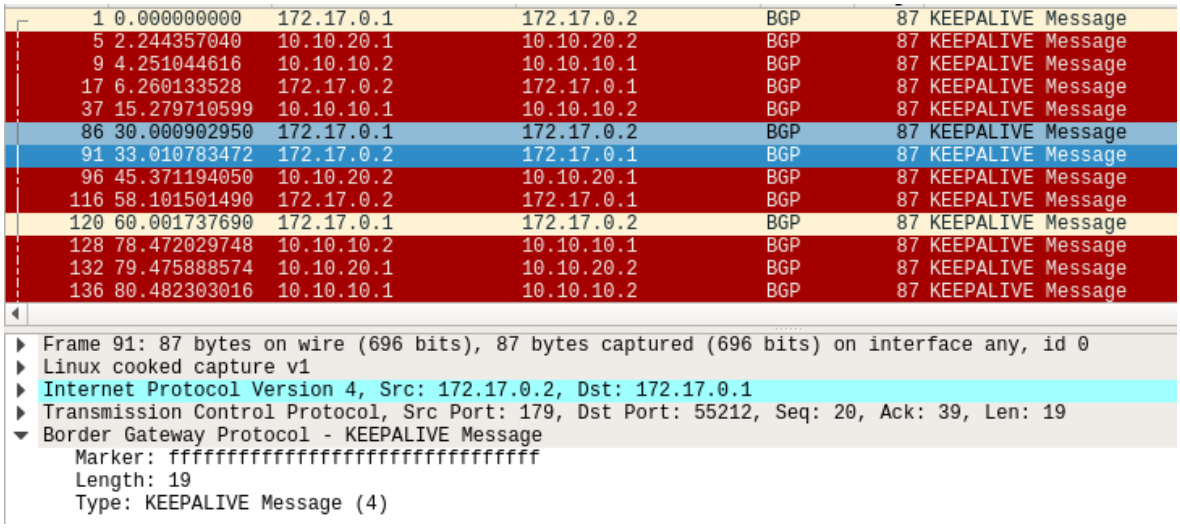


Figura A.88: Captura de paquetes mediante Wireshark

A.11.6. Preguntas y resoluciones:

Una vez realizada la práctica se deben responder las siguientes interrogantes:

- ¿Cuáles son los beneficios del plugin BGP ODL de Opendayligth?
- ¿Que tanta complejidad y utilidad cree que tiene esta implementación?

A.11.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.11.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

A.12. Práctica 12: VXLAN

A.12.1. Objetivos:

- Entender los beneficios de VXLAN.
- Conectar 2 servidores de distintos lugares usando VXLAN.
- Analizar paquetes capturados usando Wireshark.

A.12.2. Introducción:

En esta práctica se debe crear una interfaz de túnel VXLAN para comunicar dos servidores ubicados en distintos lugares.

VXLAN aborda requisitos del centro de datos de capa 2 como de capa 3, que se adaptan a múltiples inquilinos. VXLAN se ejecuta en la red existente permitiendo extender una red de capa 2. Las máquinas virtuales dentro del mismo segmento VXLAN pueden comunicarse entre sí, cada segmento VXLAN se identifica mediante un ID de 24 bits (VNI). Las máquinas virtuales en diferentes redes superpuestas VXLAN no pueden comunicarse entre ellas. VXLAN también puede ser considerado un esquema de tunelización para la superposición de redes de capa 2 sobre redes de capa 3.

El punto final del túnel (VTEP) es aquel que origina o culmina un túnel VXLAN. Este se ubica en el servidor que aloja la VM, por lo que el encapsulado del encabezado externo/túnel solo es conocido por el punto final del túnel, es decir la VM no tiene conocimiento del mismo.

En OpenFlow el campo ID de túnel es el encargado de llevar los metadatos de encapsulación asociados con un puerto lógico, al llegar un paquete a dicho puerto se usa el ID de túnel para el procesamiento interno. El ID de túnel se usa en casos como GRE, MPLS, VXLAN.

Para el desarrollo de esta práctica se requiere:

- Ordenador con alguna distribución de Sistema Operativo Linux, se recomienda Ubuntu en versiones LTS[1].
- Controlador RYU
- Mininet
- 2 máquinas virtuales con Ubuntu 18.

Esta guía se ha desarrollado considerando el software estable y compatible a la fecha (Julio 2021); el mismo que se recomienda para los apartados posteriores y se detalla en la Tabla A.12, sin embargo, se debe procurar compatibilizar e implementar la práctica con nuevas versiones en el caso de existir.

A.12.3. Marco teórico a desarrollar:

- VXLAN en SDN.
- Implementación de VXLAN en Open vSwitch

A.12.4. Instrucciones:

Para el correcto desarrollo de la presente práctica se recomienda seguir el siguiente procedimiento:

[1]Página oficial de descarga de Ubuntu: https://ubuntu.com/download/desktop

Tabla A.12: Software utilizado para el desarrollo de la guía Práctica 12

Requerimiento	Software	Versión
Sistema Operativo	Linux, Distribución Ubuntu	LTS 18.04, kernel 5.4.0
Simulador de Red SDN	Mininet	2.3.0
Virtualizador de *Switch*	Open vSwitch	2.9.8
Controlador SDN	Ryu	4.34
Lenguaje de Programación	Python	3.8.10
IDE	Spyder	3.3.6
Herramienta de Test de Tráfico	iPerf	2.0.13

1. Revisar la documentación oficial de Open vSwitch [2], que proporciona información de utilidad para el desarrollo de esta práctica.
2. **Crear la topología:** La topología de referencia a implementar se muestra en la Figura A.89, en donde el servidor Cuenca debe ser implementado en una máquina virtual y el servidor Quito en otra. Para esto se debe desarrollar *scripts* en Python, `topologia_Cuenca.py` y `topologia_Quito.py`.

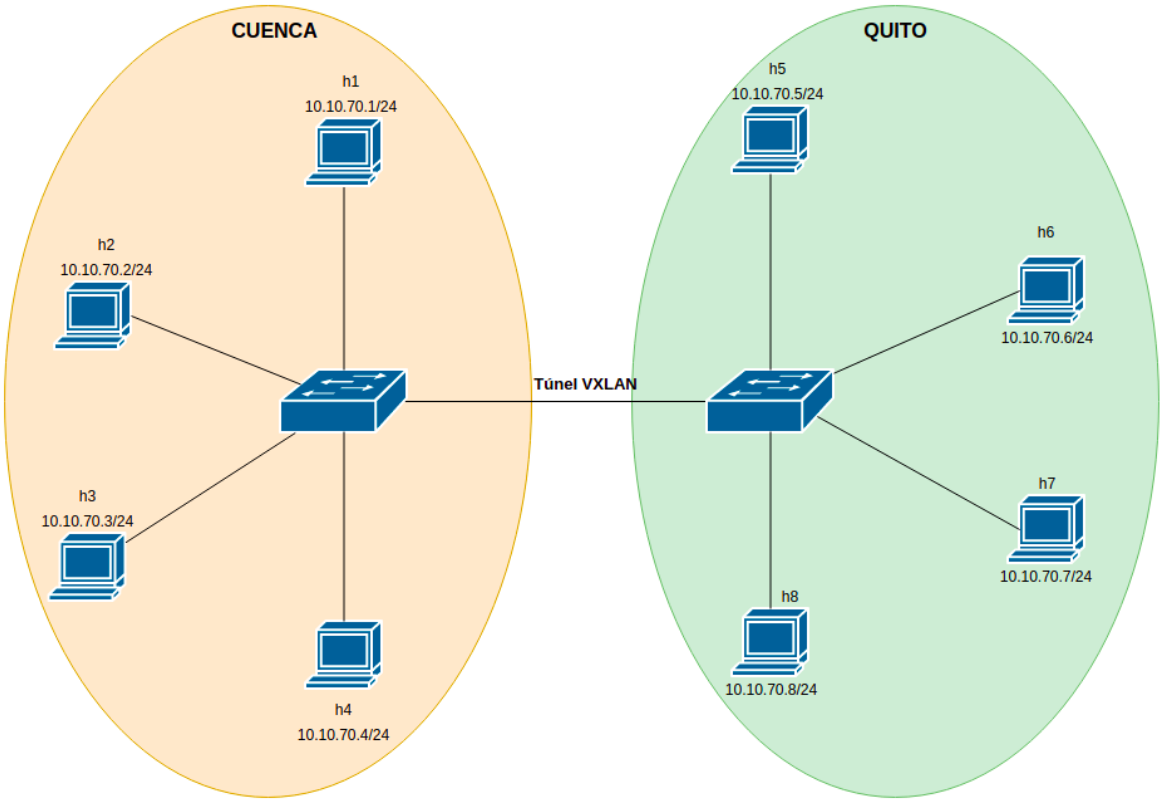


Figura A.89: Topología para VXLAN.

3. Agregar un puerto para el túnel VXLAN.
4. Realizar pruebas de conexión entre los dos servidores. (`Cuenca-Quito`)
5. Realizar capturas mediante Wireshark y analizar los resultados.
6. **Evaluar el funcionamiento:** Realizar el procedimiento de verificación de la funcionalidad de la práctica descrito en el siguiente apartado, para este fin se deberán realizar pruebas de

[2] https://docs.openvswitch.org/en/latest/howto/userspace-tunneling/

conectividad mediantes pings entre dispositivos finales pertenecientes a los distintos servidores.

A.12.5. Verificación del funcionamiento y resultados esperados:

En esta sección se detalla el procedimiento esperado para verificar el correcto funcionamiento de la práctica y se ilustran algunos de los resultados esperados. El procedimiento a seguir se resume en:

- Ejecutar el controlador.
- Ejecutar las topologías creadas.
- Verificar conectividad dentro de los servidores.
- Funcionamiento del VXLAN creado.
- Captura de paquetes mediante Wireshark.

1. En una primera máquina virtual, en un terminal se debe ubicar el directorio donde se almacena el *script* de la topología y desde ahí ejecutar la misma mediante el comando: `$sudo python topologia_Cuenca.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.90, en ésta se evidencia los enlaces entre los elementos (*switches* y *hosts*) con el detalle de los puertos involucrados.

```
mininet> net
h1 h1-eth0:SCuenca-eth1
h2 h2-eth0:SCuenca-eth2
h3 h3-eth0:SCuenca-eth3
h4 h4-eth0:SCuenca-eth4
SCuenca lo:  SCuenca-eth1:h1-eth0 SCuenca-eth2:h2-eth0
SCuenca-eth3:h3-eth0 SCuenca-eth4:h4-eth0
c1
```

Figura A.90: Salida del comando *net* en Mininet luego de ejecutar la topología.

2. En una segunda máquina virtual, abrir un terminal y ejecutar la topología con el comando: `$sudo python topologia_Quito.py`. Para comprobar la correcta creación de los enlaces y elementos de la topología, se puede digitar `net` en el CLI de Mininet. Un ejemplo de su salida se muestra en la Figura A.91.

```
mininet> net
h5 h5-eth0:SQuito-eth1
h6 h6-eth0:SQuito-eth2
h7 h7-eth0:SQuito-eth3
h8 h8-eth0:SQuito-eth4
SQuito lo:  SQuito-eth1:h5-eth0 SQuito-eth2:h6-eth0
SQuito-eth3:h7-eth0 SQuito-eth4:h8-eth0
c1
```

Figura A.91: Salida del comando *net* en Mininet luego de ejecutar la topología.

3. En el ordenador principal ejecutar el controlador con la aplicación de Ryu simple_switch_13 mediante el comando: `$sudo ryu-manager ryu.app.simple_switch_13.py`. A partir de aquí, y de manera continua, se observarán registros correspondientes a los paquetes que ingresan al

controlador (`packet in`) o errores que pueda presentar la aplicación. En la Figura A.92 se puede ver una ejecución limpia, sin errores, de la aplicación `simple_switch_13.py.py` y los registros de los primeros paquetes entrantes. En el caso de recibir errores, se deben resolver los mismos antes de continuar.

```
jhoss@jhossPC:~$ sudo ryu-manager ryu.app.simple_switch_13
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 512 00:00:00:00:00:05 33:33:00:00:00:02 2
packet in 256 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 256 00:00:00:00:00:01 33:33:ff:00:00:01 1
packet in 256 00:00:00:00:00:03 33:33:ff:00:00:03 3
packet in 256 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 256 00:00:00:00:00:02 33:33:ff:00:00:02 2
packet in 256 00:00:00:00:00:04 33:33:ff:00:00:04 4
packet in 256 00:00:00:00:00:04 33:33:00:00:00:16 4
packet in 256 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 256 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 256 00:00:00:00:00:01 33:33:00:00:00:02 1
```

Figura A.92: Ejecución de la aplicación simple_switch_13.py con el controlador Ryu y registros *packet in*

4. Se debe verificar que exista conectividad dentro de cada uno de los servidores `Cuenca-Quito`, para lo que se ejecuta el comando `pingall`, el resultado en los dos servidores debe ser similar al indicado en la Figura A.93

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura A.93: Prueba de conectividad dentro del servidor de Cuenca

5. En este punto, se verifica la conectividad entre los equipos del servidor en Cuenca y el servidor en Quito, como se evidencia en la Figura A.94, no existe conexión alguna entre los servidores.

```
mininet> h1 ping h5
ping: h5: Nombre o servicio desconocido
mininet> h1 ping 10.10.70.5 -c 3
PING 10.10.70.5 (10.10.70.5) 56(84) bytes of data.
From 10.10.70.1 icmp_seq=1 Destination Host Unreachable
From 10.10.70.1 icmp_seq=2 Destination Host Unreachable
From 10.10.70.1 icmp_seq=3 Destination Host Unreachable

--- 10.10.70.5 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, ti
me 2032ms
pipe 3
mininet> h3 ping 10.10.70.6 -c 3
PING 10.10.70.6 (10.10.70.6) 56(84) bytes of data.
From 10.10.70.3 icmp_seq=1 Destination Host Unreachable
From 10.10.70.3 icmp_seq=2 Destination Host Unreachable
From 10.10.70.3 icmp_seq=3 Destination Host Unreachable

--- 10.10.70.6 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, ti
me 2049ms
pipe 3
mininet> h4 ping 10.10.70.8
PING 10.10.70.8 (10.10.70.8) 56(84) bytes of data.
From 10.10.70.4 icmp_seq=1 Destination Host Unreachable
From 10.10.70.4 icmp_seq=2 Destination Host Unreachable
From 10.10.70.4 icmp_seq=3 Destination Host Unreachable
^C
--- 10.10.70.8 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, ti
me 4072ms
pipe 4
```

Figura A.94: Prueba de conectividad entre servidores.

6. Para que se de la conexión entre los equipos de los distintos servidores, el estudiante debe crear el túnel VXLAN, en cada uno de los *switches*, posterior a eso se vuelve a verificar la conexión entre los servidores, la salida esperada se observa en la Figura A.95.

```
mininet> h1 ping 10.10.70.5 -c 3
PING 10.10.70.5 (10.10.70.5) 56(84) bytes of data.
64 bytes from 10.10.70.5: icmp_seq=1 ttl=64 time=20.7 ms
64 bytes from 10.10.70.5: icmp_seq=2 ttl=64 time=3.30 ms
64 bytes from 10.10.70.5: icmp_seq=3 ttl=64 time=1.09 ms

--- 10.10.70.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.099/8.393/20.778/8.803 ms
mininet> h3 ping 10.10.70.6 -c 3
PING 10.10.70.6 (10.10.70.6) 56(84) bytes of data.
64 bytes from 10.10.70.6: icmp_seq=1 ttl=64 time=22.5 ms
64 bytes from 10.10.70.6: icmp_seq=2 ttl=64 time=1.59 ms
64 bytes from 10.10.70.6: icmp_seq=3 ttl=64 time=1.41 ms

--- 10.10.70.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.416/8.510/22.525/9.910 ms
mininet> h4 ping 10.10.70.8 -c 3
PING 10.10.70.8 (10.10.70.8) 56(84) bytes of data.
64 bytes from 10.10.70.8: icmp_seq=1 ttl=64 time=25.6 ms
64 bytes from 10.10.70.8: icmp_seq=2 ttl=64 time=1.88 ms
64 bytes from 10.10.70.8: icmp_seq=3 ttl=64 time=1.49 ms

--- 10.10.70.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.493/9.681/25.667/11.304 ms
```

Figura A.95: Prueba de conectividad entre servidores.

En este punto el estudiante debe ahondar en los flujos creados en cada uno de los *switches* y el por qué de los mismos.

7. En la captura de paquetes usando Wireshark se deben evidenciar los paquetes ICMP entre los equipos de cada servidor, así como la VXLAN creada.

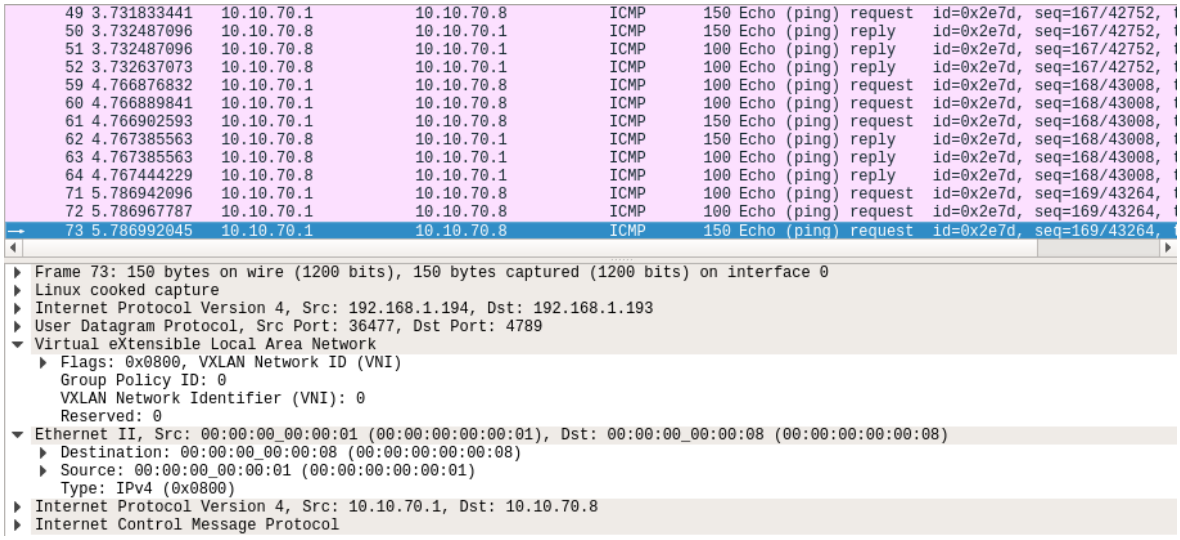


Figura A.96: Captura de paquetes mediante Wiresahrk.

A.12.6. Preguntas y resoluciones:

Una vez realizada la práctica se puede responder las siguientes interrogantes:

- ¿Cuáles son los beneficios de VXLAN?.

A.12.7. Formato del informe:

El informe debe ser presentado utilizando LaTeX con formato IEEE a una columna. Este debe contener:

- **Resumen/Abstract**
- **Introducción:** que amplíe la descripción sobre la motivación y el alcance de esta práctica.
- **Marco teórico:** de acuerdo a lo indicado previamente en A.12.3.
- **Desarrollo de la práctica:** enliste los recursos empleados, describa los pasos que se siguieron y, evidencie y analice los resultados obtenidos en cada punto.
- **Conflictos:** enumere los problemas significativos al desarrollo de la práctica y su solución o propuesta para la misma.
- **Conclusiones:** en base a los objetivos planteados, el conocimiento y la experiencia obtenida en la práctica. También busque responder a las preguntas de la sección anterior.
- **Bibliografía:** Citas en formato IEEE.

Bibliografía

[1] B. Babayiğit, S. Karakaya, y B. Ulu, "An implementation of software defined network with Raspberry Pi," in *26th IEEE Signal Processing and Communications Applications Conference, SIU 2018*. Institute of Electrical and Electronics Engineers Inc., jul 2018, pp. 1–4.

[2] D. Varnum, "OpenFlow - Teoría y conceptos básicos - / superpuesto." [En línea]. Disponible: https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/

[3] W. Stallings, F. Agboma, y S. Jelassi, "Part II Software-Defined Networks," in *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, ser. The William Stallings books on computer and data communicaitons technology. Indianápolis: Pearson, 2016, p. 538. [En línea]. Disponible: https://books.google.com.ec/books?id=Q0UOjwEACAAJ

[4] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, y M. Casado, "The design and implementation of open vSwitch," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2015*, 2015, pp. 117–130.

[5] Z. Xiang y P. Seeling, "Mininet: an instant virtual network on your computer," in *Computing in Communication Networks*, F. H. Fitzek, F. Granelli, y P. Seeling, Eds. London: Mara Conner, 2020, ch. 11, pp. 219–230. [En línea]. Disponible: https://books.google.com.ec/books?id=oXLnDwAAQBAJ&printsec=frontcover&hl=es#v=onepage&q&f=false

[6] ODCA, "OPEN DATA CENTER ALLIANCE Master USAGE MODEL: Software-Defined Networking Rev. 2.0," p. 30, 2014. [En línea]. Disponible: https://docplayer.net/2759172-Open-data-center-alliance-master-usage-model-software-defined-networking-rev-2-0.html

[7] A. Collaguazo Jaramillo, R. Alcivar, J. Pesantez, y R. Ponguillo, "Cost Effective test-bed for Comparison of SDN Network and Traditional Network," in *2018 IEEE 37th International Performance Computing and Communications Conference, IPCCC 2018*. Guayaquil: IEEE, nov 2019, pp. 1–2. [En línea]. Disponible: https://ieeexplore.ieee.org/document/8711223/

[8] J. F. Guano Viscarra, "Prototipo de una SDN utilizando herramientas Open-Source," Escuela Politécnica Nacional, Quito, Tech. Rep., 2017.

[9] O. Jarrín y D. Leonidas, "Implantación de un testbed para una red inalámbrica utilizando SDN (Open Flow)," Escuela Superior Politécnica del Ejército, Sangolquí, Tech. Rep., 2018.

[10] G. D. Guerrero Mazón, "Desarrollo de una plataforma para evaluar Calidad de Servicios (QoS) en Redes Definidas por Software (SDN)," Escuela Superior Politécnica de Chimborazo, Riobamba,

Tech. Rep., 2017. [En línea]. Disponible: http://dspace.espoch.edu.ec/bitstream/123456789/7620/1/108T0209.pdf

[11] E. M. Morales Dávila, "Integración de un IDS/IPS al controlador SDN para la prevención y Detección de Ataques de Seguridad (DoS) en un escenario de Redes Definidas por Software," Escuela Politécnica de Chimborazo, Riobamba, Tech. Rep., 2018. [En línea]. Disponible: http://dspace.espoch.edu.ec/bitstream/123456789/7620/1/108T0209.pdf

[12] S. E. Nazareno Arroyo, "Diseño e implementación de un prototipo SD-WAN basado en Raspberry Pi," Universidad de Guayaquil, Guayaquil, Tech. Rep., 2017. [En línea]. Disponible: http://repositorio.ug.edu.ec/bitstream/redug/20069/1/PROYECTODEINVESTIGACIONKIARAYABIGAILal11demayofinal.pdf

[13] C. F. Cordero Vizhñay, "Diseño y Despliegue de Funciones de Red Virtualizadas (NFV) usando Redes Definidas por Software (SDN) dentro de una infraestructura Virtual, aplicando balanceo de carga y seguridad distribuida en IPv6," Universidad Politécnica Salesiana, Cuenca, Tech. Rep., 2017.

[14] M. M. Fernández Mora y R. F. Ulloa Banegas, "Despliegue de una red SDN aplicando el protocolo MPLS y generando políticas de QoS para servicios de telefonía IP," Universidad Politécnica Salesiana, Cuenca, Tech. Rep., 2016.

[15] CEDIA, "Implementación de un Testbed SDN Empleando la Infraestructura de CEDIA," 2020. [En línea]. Disponible: https://cedia.edu.ec/es/proyectos-ganadores/cepra-vii/implementacion-testbed-sdn-empleando-infraestructura-cedia

[16] C. E. Yánez Carrera y F. G. Gallegos Pillajo, "Implementación de un Prototipo de Red Definida por Software para el Hotspot-Espoch Mediante un Controlador Basado en Openflow," 2015. [En línea]. Disponible: http://dspace.espoch.edu.ec/handle/123456789/4388

[17] C. G. Alava Rivas y D. I. Paladines Montiel, "Diseño e implementación de un módulo didáctico de red definida por software (SDN) para prácticas universitarias con protocolo Openflow mediante hardware libre," 2020. [En línea]. Disponible: http://dspace.ups.edu.ec/handle/123456789/19460

[18] G. d. J. García Rojas, "Diseño e Implementación de un Prototipo de Redes Definidas por Software (Sdn), Utilizando Mininet como Herramienta de Simulación y Físicamente con el Switch SDN ZODIAC FX," may 2020. [En línea]. Disponible: https://dspace.unl.edu.ec//handle/123456789/23231

[19] F. R. A. REYES, "ESTUDIO, DISEÑO Y SIMULACIÓN DE UNA RED DE SENSORES INALAMBRICOS (WSN) USANDO REDES DEFINIDAS POR SOFTWARE (SDN) APLICADO PARA LA UNIVERSIDAD DE GUAYAQUIL." p. 103, 2018. [En línea]. Disponible: http://repositorio.ug.edu.ec/handle/redug/32895

[20] A. E. Aragón García, *Redes definidas por software(SDN): un nuevo paradigma frente a las redes convencionales*, t. Martínez Delgadillo, Aldo René, Ed., 2020. [En línea]. Disponible: http://riul.unanleon.edu.ni:8080/jspui/handle/123456789/7724

[21] A. J. Pachés y Ó. Romero, "Estudio del controlador SDN Ryu sobre una Raspberry-Pi Model 4," p. 5, 2019. [En línea]. Disponible: www.etsit.upv.es

[22] M. Betegón, "Estudio de técnicas de Ingeniería de Tráfico basadas en SDN," Tech. Rep., 2018.

[23] D. Gonzalez, C. Mellado, K. Waltam, y A. Lara, "Low-cost SDN switch comparison: Zodiac FX and raspberry PI," in *Proceedings - 4th Jornadas Costarricenses de Investigacion en Computacion e Informatica, JoCICI 2019*. Institute of Electrical and Electronics Engineers Inc., aug 2019.

[24] ITU-T, "Y.3300: Framework of software-defined networking," 2014. [En línea]. Disponible: https://www.itu.int/rec/T-REC-Y.3300/es

[25] ONF, "Software-Defined Networking (SDN) Definition - Open Networking Foundation." [En línea]. Disponible: https://opennetworking.org/sdn-definition/

[26] M. Duarte, "Herramientas de simulación/emulación SDN," 2016. [En línea]. Disponible: https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/19028/1/2526.pdf

[27] L. Y. Becerra Sánchez, B. Valencia-Suárez, S. Santacruz-Pareja, y J. J. Padilla-Aguilar, "Uso de Mininet y Openflow 1.3 para la enseñanza e investigación en redes IPv6 definidas por software," *Revista Educación en Ingeniería*, vol. 12, num. 24, p. 89, 2017.

[28] OpenDayLight, "OpenDay light User Guide," pp. 0–16, 2011.

[29] Linux Foundation, "What is Open vSwitch?" pp. 1–2, 2016. [En línea]. Disponible: https://docs.openvswitch.org/en/latest/intro/what-is-ovs/

[30] B. Valencia y S. Santacruz, "Mininet: una herramienta versátil para emulación y prototipado de Redes Definidas por Software," *Entre ciencia e ingeniería*, vol. 9, num. 17, pp. 62–70, may 2015. [En línea]. Disponible: https://biblioteca.ucp.edu.co/ojs/index.php/entrecei/article/view/2492

[31] Mininet Project Contributors, "Mininet: An Instant Virtual Network on Your Laptop (or Other PC)." [En línea]. Disponible: http://mininet.org/

[32] Bob Lantz, "GitHub - mininet/mininet: Emulator for rapid prototyping of Software Defined Networks." [En línea]. Disponible: https://github.com/mininet/mininet

[33] M. Peuster, H. Karl, y S. Van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2016*. Palo Alto, CA, USA: Institute of Electrical and Electronics Engineers Inc., may 2017, pp. 148–153.

[34] B. Lantz y B. O'Connor, "Mininet Walkthrough - Mininet," 2021. [En línea]. Disponible: http://mininet.org/walkthrough

[35] B. D. Media, "The Complete Raspberry Pi Manual," *BDM's Ultimate Series*, p. 147, 2020.

[36] E. Upton y G. Halfacree, *Raspberry Pi® User Guide*, 4ésima ed. Wiley, 2016.

[37] Raspberry Pi Foundation, "Raspberry Pi 3 Model B+," *Raspberry Pi 3 Model B+ product brief*, pp. 1–5. [En línea]. Disponible: www.raspberrypi.org/products/raspberryhttps://datasheets.raspberrypi.org/rpi3/raspberry-pi-3-b-plus-product-brief.pdf

[38] ——, "Raspberry Pi 4 Computer Model B," *Raspberry Pi 4 Model B Product Brief*, pp. 1–6, jan 2021. [En línea]. Disponible: www.raspberrypi.orghttps://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf

[39] 11 IONOS España S.L.U., "10 sistemas operativos para Raspberry Pi - IONOS," oct 2020. [En línea]. Disponible: https://www.ionos.es/digitalguide/servidores/know-how/sistemas-operativos-para-raspberry-pi/

[40] Raspberry Pi Foundation, "Raspberry Pi OS - Raspberry Pi Documentation." [En línea]. Disponible: https://www.raspberrypi.org/documentation/raspbian/

[41] ——, "Operating system images – Raspberry Pi." [En línea]. Disponible: https://www.raspberrypi.org/software/operating-systems/

[42] Colaboradores de BIRD, "Demonio de enrutamiento de Internet Bird (Bird Internet routing daemon) - wikipe.wiki." [En línea]. Disponible: https://wikies.wiki/wiki/en/Bird_Internet_routing_daemon