UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE TELECOMUNICACIONES

INFORME FINAL DEL TRABAJO DE INTEGRACIÓN CURRICULAR, PROYECTO DE INVESTIGACÓN

TEMA:

"DESARROLLO DE UNA INFRAESTRUCTURA DE RED VIRTUAL QUE INTEGRA LAS TECNOLOGÍAS DE RED DEFINIDA POR SOFTWARE (SDN) Y VIRTUAL EXTENSIBLE LAN (VXLAN)"

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

Línea de investigación: Desarrollo, aplicación de software y cibersecurity (seguridad cibernética)

AUTOR:

Carrión Paredes Andrés Agustín

DIRECTOR:

Msc. Domínguez Limaico Hernán Mauricio

Ibarra, 2025

IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional, determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	2100981295		
APELLIDOS Y NOMBRES:	Carrión Paredes Andrés Agustín		
DIRECCIÓN:	El Olivo		
EMAIL:	aacarrionp@utn.edu.ec		
TELÉFONO FIJO:	----------	**TELF. MOVIL**	0967471234

DATOS DE LA OBRA	
TÍTULO:	"DESARROLLO DE UNA INFRAESTRUCTURA DE RED VIRTUAL QUE INTEGRA LAS TECNOLOGÍAS DE RED DEFINIDA POR SOFTWARE (SDN) Y VIRTUAL EXTENSIBLE LAN (VXLAN)"
AUTOR (ES):	Carrión Paredes Andrés Agustín
FECHA: AAAAMMDD	31 de julio de 2025
SOLO PARA TRABAJOS DE INTEGRACIÓN CURRICULAR	
CARRERA/PROGRAMA:	**X GRADO ☐ POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Telecomunicaciones
DIRECTOR:	Msc. Hernán Mauricio Domínguez Limaico
ASESOR:	Msc. Edgar Alberto Maya Olalla

AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, CARRIÓN PAREDES ANDRÉS AGUSTÍN, con cédula de identidad Nro. 2100981295 en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de integración curricular descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 144.

Ibarra, a los 31 días, del mes de Julio del 2025.

EL AUTOR:

.....

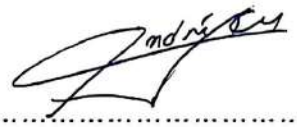
Carrión Paredes Andrés Agustín

CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 31 días, del mes de Julio del 2025.

EL AUTOR:



.....

Carrión Paredes Andrés Agustín

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE

INTEGRACIÓN CURRICULAR

Ibarra, 31 de Julio del 2025

MSC. HERNÁN MAURICIO DOMÍNGUEZ LIMAICO

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f) ... ...

Msc. Hernán Mauricio Dóminguez Limaico

C.C.: 1002379301

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificado del trabajo de Integración Curricular "DESARROLLO DE UNA INFRAESTRUCTURA DE RED VIRTUAL QUE INTEGRA LAS TECNOLOGÍAS DE RED DEFINIDA POR SOFTWARE (SDN) Y VIRTUAL EXTENSIBLE LAN (VXLAN)" elaborado por CARRIÓN PAREDES ANDRÉS AGUSTÍN, previo a la obtención del título de INGENIERO EN TELECOMUNICACIONES aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

(f):.....

Msc. Hernán Mauricio Domínguez Limaico

C.C.: 1002379301

(f):.....

Msc. Edgar Alberto Maya Olalla

C.C.: 1002702197

DEDICATORIA

Dedico este trabajo, en primer lugar, a mí mismo, por cada paso dado con determinación, por las veces que dudé y aun así continué, por creer en mis sueños incluso cuando parecían lejanos. Hoy celebro no solo un logro académico, sino un crecimiento personal profundo.

A mi padre Liborio y a mi madre Elsa, por ser faros en mi camino, por su amor silencioso y constante, por enseñarme con su ejemplo que la perseverancia y la humildad abren puertas que el miedo no se atreve a tocar.

A mis hermanas Teresa y Dayana, por su compañía incondicional, por las palabras de aliento en los días grises y por recordarme siempre quién soy y de dónde vengo.

A ustedes, mi familia, les debo más de lo que las palabras pueden decir. Este logro es también suyo, porque en cada página de esta tesis, late el amor, la paciencia y el apoyo que me han regalado.

.

AGRADECIMIENTO

Al llegar al final de esta etapa tan significativa, deseo expresar mi más profundo agradecimiento a todas las personas que hicieron posible la culminación de esta tesis y de mi formación como Ingeniero en Telecomunicaciones.

Agradezco, en primer lugar, a Dios, por darme la vida, la salud y la fortaleza necesaria para enfrentar los retos que este proceso implicó.

A mis padres, Liborio y Elsa, por su amor incondicional, su guía y su ejemplo de esfuerzo y honestidad. Gracias por enseñarme que el trabajo con integridad y el sacrificio tienen su recompensa.

A mis hermanas, Teresa y Dayana, por ser parte fundamental de mi vida, por sus palabras de ánimo y por estar siempre presentes en los momentos que más lo necesité.

A mis mentores que integran la carrera de Telecomunicaciones, por compartir sus conocimientos con compromiso y dedicación, y por inspirarme a ser cada día un mejor profesional. Su visión como orientación ha sido clave para mi desarrollo académico y humano.

A mis compañeros y amigos, con quienes compartí experiencias, desafíos y aprendizajes que marcaron profundamente mi formación. Gracias por su apoyo, compañerismo y amistad sincera.

Finalmente, agradezco a todos quienes, de una u otra forma, aportaron en este camino. Este logro es el reflejo del esfuerzo conjunto de quienes creyeron en mí y me acompañaron en este viaje.

RESUMEN EJECUTIVO

La evolución de las redes ha impulsado la adopción de tecnologías más flexibles, escalables y eficientes. En este contexto, la integración de Redes Definidas por Software (SDN) y Virtual Extensible LAN (VXLAN) se presenta como una solución innovadora para optimizar la administración y segmentación de redes virtualizadas en grandes empresas o centros de datos.

Este trabajo tiene como objetivo desarrollar una infraestructura de red virtual basada en las tecnologías SDN y VXLAN, utilizando un controlador Floodlight para la gestión centralizada y VXLAN para la segmentación de capa 2 sobre redes de capa 3. Para evaluar su desempeño, se diseña un escenario aplicando varios conceptos de redes, como la arquitectura Spine-Leaf, un modelo ampliamente utilizado en centros de datos. La implementación se realiza con el emulador de redes Mininet, permitiendo analizar el comportamiento de la red en términos de conectividad, rendimiento y eficiencia operativa.

Según los resultados observados, se demuestra que la infraestructura diseñada con SDN-VXLAN y aplicada bajo el modelo de arquitectura Spine-Leaf muestra mejores características en términos escalabilidad, automatización y flexibilidad en comparación a las redes tradicionales. Además, se realiza un análisis sobre el comportamiento de la red bajo diferentes condiciones, evaluando la capacidad de respuesta y eficiencia.

En definitiva, es evidente que el trabajo con SDN y VXLAN constituye una solución eficaz para la virtualización y gestión de redes por las características que ofrece, proporcionando una base sólida que mejorará las optimizaciones en infraestructuras a gran escala, como centros de datos y redes empresariales.

Palabras clave: SDN, VXLAN, Floodlight, virtualización de red, Mininet, VLAN, Spine-Leaf.

ABSTRACT

The evolution of networks has driven the adoption of more flexible, scalable, and efficient technologies. In this context, the integration of Software-Defined Networking (SDN) and Virtual Extensible LAN (VXLAN) emerges as an innovative solution to optimize the management and segmentation of virtualized networks in large enterprises or data centers.

This work aims to develop a virtual network infrastructure based on SDN and VXLAN technologies, using a Floodlight controller for centralized management and VXLAN for Layer 2 segmentation over Layer 3 networks. To evaluate its performance, a scenario is designed applying various networking concepts, such as the Spine-Leaf architecture, a model widely used in data centers. The implementation is performed using the Mininet network emulator, allowing for analysis of network behavior in terms of connectivity, performance, and operational efficiency.

According to the observed results, it is demonstrated that the infrastructure designed with SDN-VXLAN and implemented under the Spine-Leaf architecture model exhibits superior characteristics in terms of scalability, automation, and flexibility compared to traditional networks. In addition, an analysis is performed on network behavior under different conditions, evaluating its responsiveness and efficiency. Ultimately, it's clear that working with SDN and VXLAN is an effective solution for network virtualization and management due to its features, providing a solid foundation that will improve optimizations in large-scale infrastructures such as data centers and enterprise networks.

Keywords: SDN, VXLAN, Floodlight, network virtualization, Mininet, VLAN, Spine-Leaf.

LISTA DE SIGLAS

SDN. Software Defined Networking (Red Definida por Software)

VXLAN. Virtual Extensible LAN

VLAN. Virtual Local Area Network

OpenFlow. Protocolo utilizado en redes definidas por software

Floodlight. Controlador SDN de código abierto basado en OpenFlow

Mininet. Emulador de redes definido por software

IPv4. Internet Protocol version 4

ICMP. Internet Control Message Protocol (Protocolo de Mensajes de Control de Internet)

PING. Packet Internet Groper (Herramienta para diagnóstico de conectividad de red)

UDP. User Datagram Protocol (Protocolo de Datagramas de Usuario)

TCP. Transmission Control Protocol (Protocolo de Control de Transmisión)

IPERF. Herramienta para la medición del ancho de banda y rendimiento de red

OpenFlow. Protocolo utilizado en redes definidas por software

VM: Máquina virtual

MSDC: Centros de datos a gran escala

Contenido

1 CAPITULO I. ANTECEDENTES 20

1.1 Problema de investigación. 20

1.2 Justificación 22

1.3 Objetivos 24

1.3.1 Objetivo General 24

1.3.2 Objetivos Específicos 24

2 CAPITULO II. Estado del arte 25

2.1 SDN 25

2.1.1 SDN Capa de aplicación 27

2.1.2 SDN Capa de control 29

2.1.3 SDN Capa de datos 30

2.2 Controlador SDN 31

2.3 VXLAN 32

2.3.1 Direccionamiento de VXLAN 34

2.3.2 Formato de paquete VxLAN 34

2.4 Mininet 37

2.4.1 Características de Mininet 38

3 CAPITULO III. Escenario para Implementación 39

3.1 Requerimientos para instalación 39

3.2 Escenario propuesto 41

3.3 Levantamiento de Topología sobre Python 50

3.4 Importancia del protocolo OpenFlow 56

4 CAPITULO IX: Resultados 61

4.1 OpenFLow 61

4.2 Pruebas de convergencia 69

4.2.1 Validación VXLAN con tráfico ICMP 72

4.2.2 Encapsulamiento VXLAN dentro de la Red SDN. 72

4.2.3 Diagrama de funcionamiento 76

4.2.4 Paquetes Perdidos 80

4.3 Evaluación del Rendimiento y Conectividad en la Infraestructura SDN-VXLAN 81

4.3.1 PRUEBA 1: Simulación de tráfico TCP con eventos temporizados. 85

4.3.2 PRUEBA 2: Medición de RTT (Tiempo de ida y vuelta) 91

4.3.3 PRUEBA 3: Medición de rendimiento (Throughput). 96

4.3.4 PRUEBA 4: Análisis del impacto del balanceo de carga 102

4.3.5 PRUEBA 5: Simulación de Congestión y Análisis de Laberio 106

Conclusiones 112

Recomendaciones 114

BIBLIOGRAFÍA 116

Anexo 119

Instalación y configuración de las herramientas para el diseño. 119

Instalación del Sistema Operativo 119

Instalación del controlador 127

Instalación de MININET 133

Código Topología VXLAN – SDN 136

Código Topología VXLAN – SDN – SPINE LEAF 138

Código para escenario de pruebas (VxLAN – SDN – SPINE LEAF) 141

ÍNDICE DE TABLAS

Tabla 1 *Tipos de controladores SDN.* 32

Tabla 2 *Requisitos para instalación de MININET.* 39

Tabla 3 *Requerimiento para instalación de controlador SDN Floodlight.* 40

Tabla 4 *Requerimientos para instalación de Sistema Operativo.* 41

Tabla 5 *Tabla de direccionamiento del escenario en IPv4.* 46

Tabla 6 *Parámetros definidos para pruebas implementadas.* 82

Tabla 7 *Condiciones para el proceso de simulación de tráfico TCP.* 86

Tabla 8 *Descripción de los parámetros para prueba de tráfico TCP progresiva.* 86

Tabla 9 *Datos extraídos de Wireshark de la simulación de tráfico TCP por intervalo de tiempo.* 87

Tabla 10 *Parámetros del comando para prueba RTT.* 92

Tabla 11 *Valores de RTT por segundo para cada host.* 94

Tabla 12 *Descripción de los parámetros para la prueba de Throughput.* 97

Tabla 13 *Valores de Throughput por segundo para cada host expresados en Gbps.* 100

Tabla 14 *Descripción de los parámetros para ejecución de prueba de balance de carga.* ... 103

Tabla 15 *Descripción de parámetros para prueba de congestión de enlace.* 107

Tabla 16 *Resultados obtenidos del incremento porcentual mejorada.* 110

Tabla 17 *Criterios clave para trabajar con Floodlight.* 131

ÍNDICE DE FIGURAS

Figura 1 *Arquitectura SDN.* 26

Figura 2 *SDN plano de Aplicación.* 28

Figura 3 *Estructura común de un controlador en una red SDN.* 31

Figura 4 *Formato de Paquete VXLAN.* 35

Figura 5 *Escenario estructural base (SDN – VXLAN).* 47

Figura 6 *Escenario propuesto (SDN – VxLAN – Spine Leaf).* 49

Figura 7 *Construcción de la Topología en Mininet por medio del script desarrollado en Python.* 51

Figura 8 *Configuración de túneles VXLAN.* 52

Figura 9 *Reglas agregadas para Openflow.* 53

Figura 10 *Ejecución de script con las configuraciones de red SDN con soporte VXLAN.* 54

Figura 11 *Estado del controlador Floodlight y módulos cargados.* 55

Figura 12 *Dashboard del panel de control del controlador Floodlight.* 56

Figura 13 *Topología de red visualizada en el controlador Floodlight.* 56

Figura 14 *Componentes principales de un conmutador OpenFlow.* 58

Figura 15 *Paquete Hello.* 62

Figura 16 *Paquete Features Request.* 63

Figura 17 *Paquete Features Reply.* 64

Figura 18 *Paquetes Packet Flow Mod.* 65

Figura 19 *Paquetes Packet In.* 66

Figura 20 *Paquetes Out.* 66

Figura 21 *Paquetes Multipart Request.* 67

Figura 22 *Paquetes Multipart Reply.* 68

Figura 23 *Paquetes Config Reply y Config Request.* 68

Figura 24 *Paquetes Config Reply.* 69

Figura 25 *Ejecución de script para levantamiento de red.* 69

Figura 26 *Prueba de convergencia entre todos los hosts de la red implementada.* 70

Figura 27 *Prueba de ping del equipo h1 (vtep 1) a h5 (vtep 2).* 71

Figura 28 *Prueba de ping del equipo h7 (vtep 2) a h12 (vtep 3).* 71

Figura 29 *Prueba de ping del equipo h1 (vtep 1) a h9 (vtep 3).* 71

Figura 30 *Paquetes ICMP entre hosts de distintos VTEPs.* 72

Figura 31 *Encapsulamiento y desencapsulamiento del Paquete VXLAN.* 73

Figura 32 *Original L2 Frame (Capa 3 interna).* 73

Figura 33 *Header VXLAN – Tráfico VXLAN en Red eXtensible.* 75

Figura 34 *UDP Header.* 76

Figura 35 *Outer IP header (capa 3 Underlay).* 76

Figura 36 *Flujo de comunicación entre hosts del sistema.* 77

Figura 37 *Encapsulamiento de VTEP 1 a VTEP 2.* 78

Figura 38 *Encapsulamiento de VTEP 2 a VTEP 1.* 78

Figura 39 *Encapsulamiento de VTEP 3 a VTEP 2.* 78

Figura 40 *Encapsulamiento de VTEP 2 a VTEP 3.* 79

Figura 41 *Encapsulamiento de VTEP 1 a VTEP 3.* 79

Figura 42 *Encapsulamiento de VTEP 3 a VTEP 1.* 79

Figura 43 *Paquete ICMP sin Respuesta.* 80

Figura 44 *Carpeta de resultados generados por las pruebas realizadas.* 85

Figura 45 *Metodología para la simulación de tráfico TCP progresivo.* 85

Figura 46 *Representación de la evolución del tráfico TCP en función del tiempo.* 90

Figura 47 *Metodología para simulación de la prueba 2.* 91

Figura 48 *Evidencia de los resultados RTT en el archivo txt.* 92

Figura 49 *Resultados de la simulación de la prueba 2 – Evaluación de RTT.* 94

Figura 50 *RTT promedio obtenido en la prueba 2.* 96

Figura 51 *Metodología para simulación de la prueba 3.* 97

Figura 52 *Evidencia de los resultados del throughput en el archivo txt.* 98

Figura 53 *Resultados de la simulación de la prueba 3.* 100

Figura 54 *Metodología para la simulación de la prueba 4.* 102

Figura 55 *Evidencia de las pruebas de balanceo en el txt para s3.* 103

Figura 56 *Evidencia de las pruebas de balanceo en el txt para s4.* 104

Figura 57 *Evidencia de las pruebas de balanceo en el txt para s5.* 104

Figura 58 *Resultados de Balanceo de Carga total para cada equipo.* 105

Figura 59 *Metodología para simulación de prueba 5.* 107

Figura 60 *Evidencia de la captura antes de la congestión.* 108

Figura 61 *Evidencia de la captura después de la congestión.* 108

Figura 62 *Resultados de prueba de congestión en enlace.* 109

Figura 63 *Configuración del nombre y sistema operativo de la VM.* 119

Figura 64 *Configuración del hardware de la máquina virtual.* 120

Figura 65 *Creación del disco duro virtual.* 121

Figura 66 *Resumen de la configuración de la máquina virtual.* 121

Figura 67 *Pantalla de bienvenida de la instalación de Ubuntu.* 122

Figura 68 *Preparación para la instalación de Ubuntu.* 123

Figura 69 *Selección del tipo de instalación de Ubuntu.* 124

Figura 70 *Confirmación de los cambios en el disco.* 124

Figura 71 *Configuración de la zona horaria.* 125

Figura 72 *Configuración de la disposición del teclado.* 126

Figura 73 *Configuración del usuario principal.* 126

Figura 74 *Instalación de herramienta Git en el sistema.* 127

Figura 75 *Instalación de componente ANT.* 128

Figura 76 *Instalación OpenJDK 8.* 128

Figura 77 *Descarga del archivo tar.gz del controlador Floodlight.* 129

Figura 78 *Descompresión y ubicación de la carpeta Floodlight-1.2.* 129

Figura 79 *Iniciación del controlador SDN.* 130

Figura 80 *Interfaz del controlador desde el navegador.* 131

Figura 81 *Clonación del repositorio oficial de MININET.* 133

Figura 82 *Verificación de la versión de Python disponible en el sistema operativo.* 134

Figura 83 *Ejecución de script para instalación de MININET y sus componentes.* 134

Figura 84 *Interfaz gráfica de MININET.* 135

1 CAPITULO I. ANTECEDENTES

1.1 Problema de investigación.

Las redes definidas por software (SDN) son una manera de abordar la creación de redes en la cual el control se desprende del hardware y se le da el mismo a una aplicación de software llamada controlador. El término SDN (Software Defined Network o red definida por software) se ha venido acuñando en los últimos años para hacer referencia a una arquitectura de red que permite separar el plano del control, del plano de datos, para conseguir redes más programables, automatizables y flexibles. Con SDN se virtualiza la red independizándola de la infraestructura física subyacente (Norberto Figuerola, 2014).

Desde otro punto de vista, según (Amaya Fariño et al., 2022), en la era digital, las redes IP tradicionales aún son complicadas y tediosas de administrar. Existe la dificultad para configurar la red de acuerdo con los procedimientos predefinidos y responder a las modificaciones de carga y fallas a través de la reconfiguración de la red. Están integradas verticalmente para complicar mucho más las cosas: los planos de control y datos están agrupados juntos, por esa razón las empresas de telecomunicaciones han creado la evolución en comunicación de datos, conocidas como SDN o redes manipuladas por software donde no es necesario el modelo jerárquico de capas clásico modelo OSI.

VXLAN o LAN virtual extensible, es un protocolo estandarizado que según (*RFC7348*, 2014) emplea un esquema de superposición MAC sobre IP/UDP incrementando el número de redes de capa 2 admitidas de 4096 a 16 millones. VXLAN fue diseñada principalmente para abordar los crecientes requisitos de escalabilidad de los MSDC permitiendo que los segmentos de Capa 2 se extiendan por todo el CPD (Data processing center) e incluso entre CD (Compact Disc). Esto es posible gracias a la superposición de redes virtuales de Capa 2 sobre redes de Capa 3 subyacentes, básicamente, superposición basada en máquina. No obstante, VXLAN

también se puede utilizar como una superposición basada en red ya que brinda ventajas de escalabilidad a los switches Leaf en una arquitectura Fabric o Clos. En los centros de datos, VXLAN es el protocolo más utilizado para crear redes superpuestas. El protocolo VXLAN aborda las necesidades de los centros de datos de múltiples inquilinos al proporcionar la segmentación necesaria a gran escala.

La creciente evolución de las redes de comunicación junto con la diversificación de aplicaciones y servicios en entornos de centros de datos y redes empresariales han generado nuevos desafíos. Los sistemas tradicionales de redes basadas en hardware y la segmentación mediante VLANs (Virtual Local Area Networks) no son suficientes para satisfacer las necesidades de infraestructuras escalables, flexibles y eficientes en un mundo de rápido cambio tecnológico. La problemática se centra en la incapacidad para gestionar de manera efectiva la creciente demanda de segmentación de red y aislamiento de aplicaciones y servicios en un entorno que requiere mayor dinamismo y gran escala (Pérez E & Molina K, 2023).

La integración de tecnologías como SDN (Software-Defined Networking) con VXLAN (Virtual Extensible LAN) permite crear redes virtuales superpuestas que ofrecen ventajas clave en términos de segmentación, escalabilidad, movilidad de máquinas virtuales, flexibilidad en la asignación de recursos y optimización de rendimiento. SDN se encarga de ofrecer una gestión centralizada y dinámica, mientras que VXLAN hace posible extender la infraestructura de red, permitiendo que las redes virtuales rompan límites tradicionales de la red física. Gracias a esta combinación, se mejora entornos empresariales, centros de datos y nubes, donde se requiere soluciones que involucran adaptabilidad, seguridad y administración eficiente de recursos (Cobos Yilder, 2021).

1.2 Justificación

Este proyecto se justifica en vista de la creciente importancia de las tecnologías, puesto que, el mundo de las redes está experimentando cambios a un ritmo sorprendentemente rápido, y el futuro de las redes definidas por software (SDN) es sumamente prometedor. La capacidad de programar y automatizar una red abre un abanico de oportunidades tanto para la industria como para los profesionales de TI (Tecnologías de la información). La introducción de otras tecnologías, como el Big Data, Internet de las cosas o la Inteligencia Artificial, secundan la necesidad de redirigir las redes hacia un nuevo plano de control que sea independiente del hardware del fabricante (Castro Serantes, 2023). En otras palabras, las redes tradicionales están siendo reemplazadas con nuevas tecnologías, para reducir costes y centralizar la gestión, por tal motivo las empresas se ven forzadas a buscar nuevas soluciones para satisfacer la demanda de recursos (Aguirre F. & Crespo M., 2021).

Las Redes Definidas por Software (SDN, por sus siglas en inglés), surgen como una opción para las nuevas arquitecturas de red; una definición aceptable de las redes SDN, es, separar el plano de datos y el plano de control, para ser todo controlado por medio de una aplicación que se denomina controlador. Además, las redes no tienen manipulación por hardware, esto quiere decir, que ya no se debe configurar cada dispositivo y todo pasa a ser controlado por una aplicación, ya sea en un ordenador o en algún dispositivo móvil. Gracias a ello se soluciona los problemas de las redes actuales, dado que, el plano de control está situado en un servidor central que se encargará de configurar y gestionar todos los equipos de la infraestructura de red, ahorrando en personal costoso y sobre todo en tiempo de configuración de una red (Salazar, 2021), lo que pretende este tipo de redes, es separar el plano de control (software) del plano de datos (hardware).

En la actualidad, las redes SDN, están en crecimiento por sus amplias ventajas con respecto a las demás arquitecturas, muchas empresas, principalmente las que se encuentran en

evolución, buscan mejorar su oferta de negocios, implementando nuevos servicios; no obstante, operadoras de gran prestigio como son GOOGLE, FACEBOOK, YAHOO!, están migrando sus redes actuales, a redes definidas por software, dado que afrontan cada día, miles de nuevos usuarios solicitando sus servicios de internet. Un gran problema con el que estas operadoras deben lidiar es la escalabilidad de sus redes. Cuando un usuario va a realizar una búsqueda de información en el portal de Google, el intercambio de datos entre los nodos de esta red puede llegar a los 1000 Terabytes. Aquí radica la importancia de implementar las redes SDN, que brindarán mayor escalabilidad, proporcionando un eficiente rendimiento y mejorando la conectividad entre los cientos de miles de usuarios (Leonardo et al., 2015). Las redes definidas por software (SDN), son un modelo es altamente escalable debido que permite y admite muchas tecnologías actuales independientemente de los fabricantes y está pensado para lograr una red más administrable y sencilla de entender, también es de bajo costo debido a que el enfoque principal es la virtualización de hardware en la nube (Underdahl Brian & Kinghorn Gary, 2015).

Con este estudio se busca ofrecer una visión clara y comprensión profunda de las ventajas y desafíos asociados con la implementación conjunta de SDN y VXLAN en redes de mayor escala, como las que se encuentran en empresas y universidades (VMWare, 2023). Considerando la importancia actual de la tecnología de redes en el proceso de transformación digital y la continua evolución de las demandas de comunicación, esta investigación contribuirá un conocimiento relevante sobre el impacto de estas tecnologías al potenciar la infraestructura de red en organizaciones de diferentes sectores, abriendo la posibilidad de promover la eficacia y la competitividad en las organizaciones.

1.3 Objetivos

1.3.1 Objetivo General

Implementar una infraestructura de red virtual que integre las tecnologías de Red Definida por Software (SDN) y Virtual Extensible LAN (VXLAN), con el propósito de mejorar la entrega de servicios de red.

1.3.2 Objetivos Específicos

1. Realizar un estudio del estado del arte sobre las tecnologías SDN y VXLAN para la obtención de aspectos tales como funcionamiento, características y tecnología en general.
2. Establecer y emular un escenario combinado entre SDN y VXLAN mediante el empleo de un software de diseño de redes de código abierto para la demostración funcional entre las tecnologías.
3. Desplegar una infraestructura virtual basada en la metodología de cascada, estableciendo los pasos a seguir en cada una de las fases, cumpliendo con los objetivos planteados.
4. Evaluar los resultados obtenidos al implementar una red que aborde las tecnologías SDN y VXLAN, identificando de manera precisa y documentada los impactos positivos y beneficios que aportan las tecnologías mencionadas en una infraestructura de red.

2 CAPITULO II. ESTADO DEL ARTE

En este capítulo se presenta la teoría que brindará el conocimiento sustancial para lograr el entendimiento de las tecnologías y el escenario.

2.1 SDN

Mirando hacia el pasado, se observa que los dispositivos de hardware no han experimentado un desarrollo significativo, a pesar de los avances constantes en las capacidades de la estructura de conmutación, las velocidades de interfaz y las comunicaciones de datos desde la introducción de tecnologías como IP, MPLS[1] o móviles. IP y MPLS posibilitaron que los operadores de red establecieran redes y superposiciones virtuales sobre infraestructuras base, de manera similar a cómo los operadores de centros de datos pudieron crear máquinas virtuales sobre las máquinas físicas con la llegada de la virtualización. Hace alrededor de una década, surgió un nuevo paradigma en el ámbito de las redes conocido como Redes Definidas por Software (SDN), que ofrecía características innovadoras capaces de superar las limitaciones existentes en los dispositivos de hardware (Castro Serantes, 2023).

Según (Paucar A, 2022), SDN es una arquitectura diseñada para abastecer el ritmo de crecimiento de las aplicaciones en la actualidad, su funcionamiento se basa en separar la administración de la red de la infraestructura de red subyacente, para así conseguir un flujo de tráfico más dinámico, con ello reducir la complejidad de las redes definidas estáticamente, automatice las funciones de la red y simplifiquen la implementación.

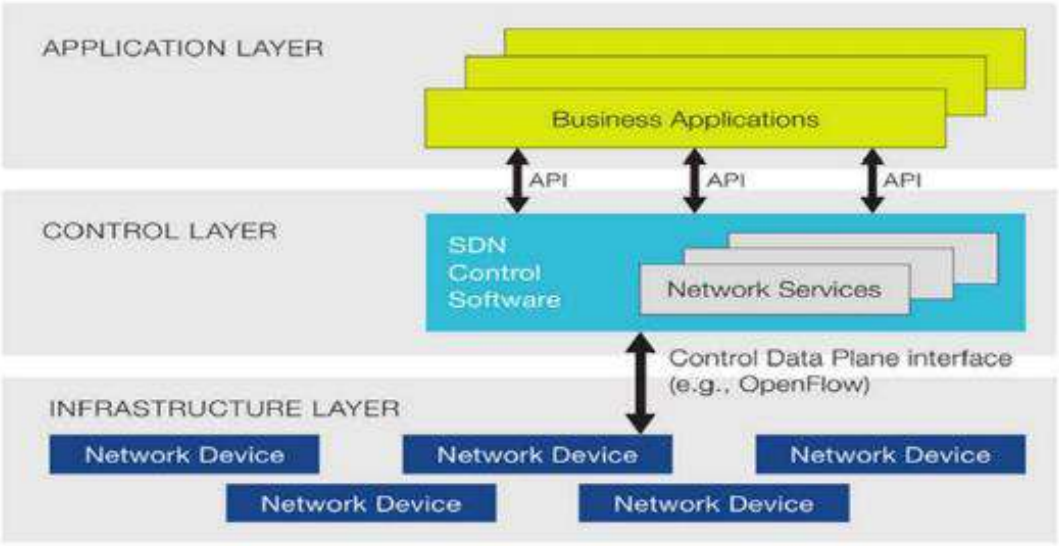
La Open Networking Foundation (ONF) ha introducido una arquitectura de referencia para SDN que sigue un modelo de tres capas, tal como se muestra en la Figura 1. El propósito principal de esta arquitectura es proporcionar una visión general de alto nivel de los puntos de referencia y las interfaces abiertas que deben estar presentes en cualquier implementación de

[1] MPLS (Multiprotocol Label Switching): Tecnología de enrutamiento que utiliza etiquetas en lugar de direcciones IP para agilizar el tráfico en redes de alto rendimiento.

SDN. Esto garantiza un conjunto mínimo de capacidades esenciales para supervisar la conectividad ofrecida por los recursos de red y dirigir el flujo de tráfico a través de ellos. La meta subyacente de esta estructura es establecer un marco común que facilite la interoperabilidad y promueva la adopción más extensa de SDN en diversos entornos (Mendiola et al., 2017).

Figura 1

Arquitectura SDN.



Nota. Recuperado de (Paucar A, 2022)

A diferencia de las redes tradicionales, las redes definidas por software (SDN) poseen la característica de separar el plano de control del plano de datos en los dispositivos de red. Esta estructura permite delegar la gestión del tráfico a un controlador central en la red. El controlador asume la función de "cerebro" de la red SDN, encargándose de las decisiones de enrutamiento y control que anteriormente estaban descentralizadas en los propios dispositivos de red en las arquitecturas tradicionales.

2.1.1 SDN Capa de aplicación

Las aplicaciones en el ámbito de las Redes Definidas por Software (SDN) comunican sus requisitos a la red a través de una interfaz de programación de aplicaciones (API[2]) conectada a la capa de control. Estas aplicaciones tienen como objetivo satisfacer las necesidades de los usuarios y forman la capa de aplicación, que comprende las aplicaciones de usuario. Los controladores SDN ponen a disposición una interfaz estandarizada que permite desarrollar aplicaciones de manera más flexible y facilita la comunicación entre estas aplicaciones y el propio controlador.

En la actualidad, es común que los controladores de red incluyan APIs Northbound, como APIs RESTful, sistemas de archivos y lenguajes de programación. Las aplicaciones abarcan funciones como enrutamiento, cortafuegos, balanceadores de carga, seguimientos, entre otras. En esencia, una aplicación de gestión establece políticas que, en última instancia, se traducen en instrucciones específicas para la interfaz SouthBand, programando así el comportamiento de los dispositivos de reenvío (Rodríguez Herlein et al., 2020).

Un ejemplo ilustrativo del empleo de APIs RESTful se evidencia en el trabajo desarrollado por (Gonzáles Joan, 2023), titulado "Reconocimiento de patrones en métricas de tráfico para el cumplimiento de los SLA (Service Level Agreement) en entornos de redes B5G (Beyond 5G)". En este estudio, se implementa una API REST para la interacción con el sistema y realización de pruebas que evalúen su desempeño. Es importante destacar que las APIs RESTful están diseñadas con puntos finales que se adhieren a los principios de la arquitectura REST, permitiendo la interacción con servicios web de manera eficiente. El plano de aplicación engloba múltiples aplicaciones de red diseñadas para la administración y control específicos de la red. Es esencial señalar que no existe un conjunto preciso de estas aplicaciones, ni siquiera

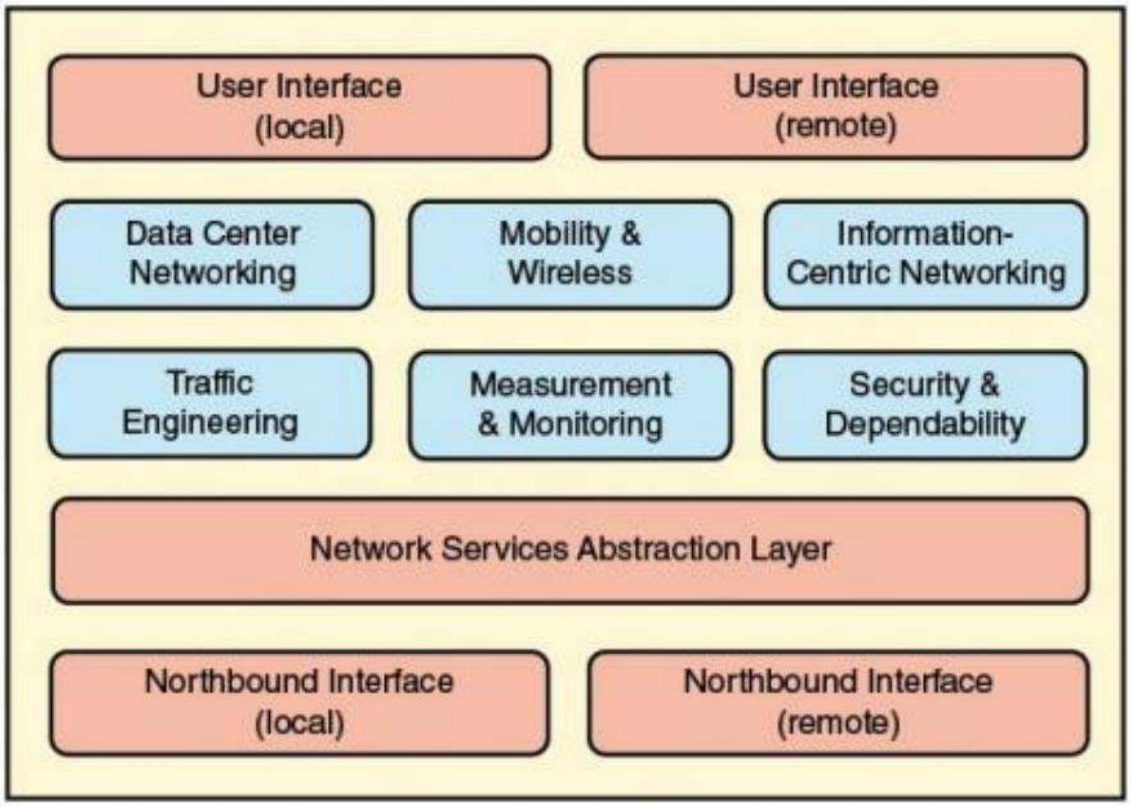
[2] API (Application Programming Interface): Conjunto de reglas y herramientas que permiten la comunicación entre diferentes software o sistemas.

una lista específica. De esta manera, la capa de aplicación puede incorporar servicios y herramientas de red generales que son implementados y pueden considerarse como parte esencial de la funcionalidad del plano de control (Stallings, 2015).

En la **Figura 2** se tiene una visión general del plano de aplicación en SDN, presentando elementos y áreas de aplicación específicas con un enfoque de abajo hacia arriba. Aquí, se muestran interfaces de dirección norte (Northbound Interface), ya sean locales o remotas, utilizadas para ejecutar aplicaciones SDN en un mismo servidor dentro del plano de control. Básicamente, la interfaz de dirección norte se convierte en un protocolo o interfaz de programación de aplicaciones (API) que conecta varias aplicaciones al sistema operativo de red del controlador (NOS) (Stallings, 2015).

Figura 2

SDN plano de Aplicación.



Nota. Recuperado de (Stallings, 2015).

2.1.2 SDN Capa de control

La capa intermedia se compone del Controlador SDN, el cual posee una visión completa de la red. Siguiendo la analogía con un sistema operativo convencional, la plataforma de control simplifica los detalles de bajo nivel relacionados con la conexión y la interacción con los dispositivos de reenvío, con el propósito de concretar las políticas de red (Rodríguez Herlein et al., 2020).

En una arquitectura SDN, el controlador ocupa un papel clave al manejar tanto la capa de aplicación como la infraestructura mediante interfaces conectadas a cada uno de los planos involucrados. Su función principal es proporcionar el respaldo necesario para la lógica de control, generando la configuración de la red conforme las políticas definidas. Al interactuar con el plano de infraestructura, el controlador se encarga del estado de la red y actualiza las reglas de reenvío en los dispositivos, asegurándose de que todo esté alineado con lo que requieren las aplicaciones (Rodríguez Herlein et al., 2020).

En relación con la comunicación con las aplicaciones SDN, el controlador utiliza un lenguaje de alto nivel, con opciones que incluyen lenguajes existentes (Python, Java, C++), librerías en un kit de software de desarrollador (SDK) como OnePK de Cisco, o lenguajes emergentes como Flowbased Management Language (FML), Frenetic o Nettle. A pesar de que el modelo del plano de control es centralizado, su implementación puede distribuirse para lograr escalabilidad y redundancia (Rodríguez Herlein et al., 2020).

Es crucial que un controlador pueda comunicarse con otros, ya que depender únicamente de un único controlador puede generar problemas de congestión, como cuellos de botellas, convirtiéndolo en un punto de falla. Una de las soluciones más destacadas para lograrlo es HyperFlow, que garantiza una vista sincronizada y coherente entre varios controladores (Rodríguez Herlein et al., 2020).

2.1.3 SDN Capa de datos

La capa inferior, conocida como capa de infraestructura, comprende los elementos de red como hosts, switches/routers físicos o virtuales, y los medios de transmisión. En el plano de datos, se suministra el servicio fundamental de reenvío de datos. Se emplea el término "conmutador SDN" para referirse al componente de reenvío en este plano, aclarando que este término no restringe las funciones del dispositivo de red, ya sea un enrutador, conmutador o cortafuegos. Estos dispositivos capturan el estado de la red, que incluye la topología o las estadísticas de tráfico, y lo transmiten al controlador. Este, a su vez, les indica las reglas de reenvío de paquetes (Rodríguez Herlein et al., 2020).

Este es un punto de cambio importante respecto a las redes tradicionales. En SDN, son las aplicaciones las que definen cómo se usará la red y le envían esta información al controlador a través de las APIs Northbound. El controlador, al mismo tiempo, toma decisiones necesarias y las comunica a la infraestructura de red mediante las APIs Southbound. Estas APIs pueden ser abiertas o ser propietarias (Rodríguez Herlein et al., 2020).

Los dispositivos que se ubican en el plano de datos son capaces de soportar las siguientes funciones (Stallings, 2015):

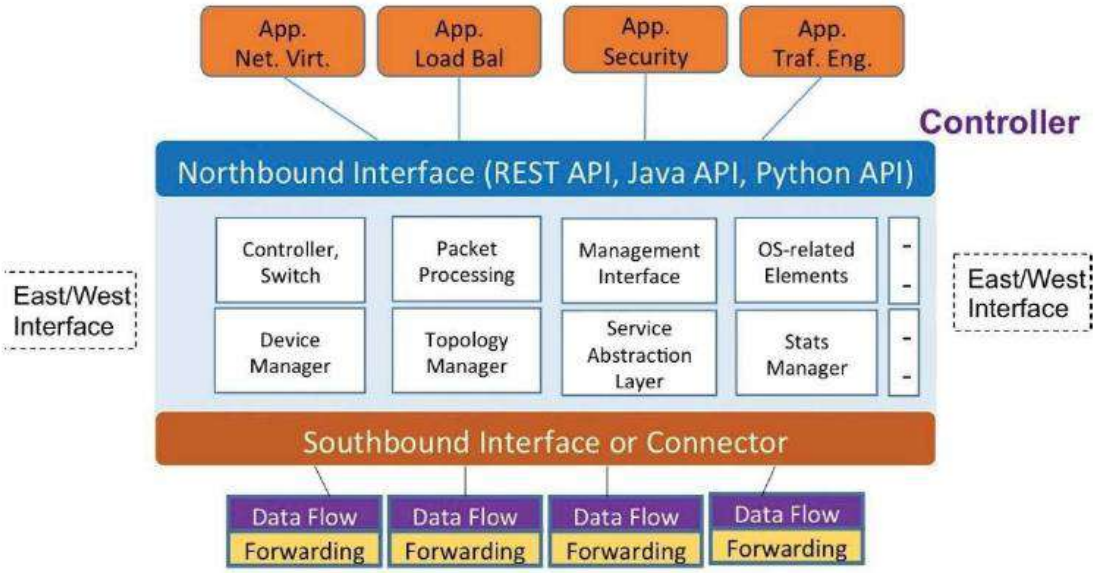
- **La función de soporte de control:** se encarga de respaldar la capacidad para programar las funciones de la capa de recursos mediante la interfaz de control de recursos.
- **Función de reenvío de datos:** se ocupa de la administración de los flujos de datos que deben ser transmitidos a lo largo de las diversas rutas establecidas de acuerdo con las necesidades de las aplicaciones de SDN. Esta capacidad es suministrada por la capa de control SDN, lo que posibilita reducir la funcionalidad de reenvío de datos en la capa de recursos de la red.

2.2 Controlador SDN

Dentro de la arquitectura de redes SDN, los controladores incorporan inteligencia mediante software para lograr una gestión completa de la red. Esta estructura se compone de tres capas: Infraestructura, Control y Aplicaciones. El propósito es proporcionar a los administradores de red un mayor control sobre los dispositivos de red al separar el plano de datos del plano de control, donde se establecen aspectos como prioridades, direccionamiento y calidad de servicio. (Manuel et al., 2014). En resumen, la idea central que consiste en centralizar los elementos de la red a través del controlador, como se presenta en la **Figura 3**.

Figura 3

Estructura común de un controlador en una red SDN.



Nota. Recuperado de (Ynga et al., 2020).

De manera general, se describe un controlador SDN como un sistema basado en software o conjunto de sistemas que, en comparación, tiene la capacidad de ofrecer diversos servicios para llevar a cabo tareas específicas en los dispositivos que componen la red definida por software. En algunos casos, este controlador también puede integrar dispositivos que no

forman parte del canal de comunicación OpenFlow. A continuación, en la **Tabla 1** se presenta algunos tipos de controladores SDN apropiados para la práctica.

Tabla 1

Tipos de controladores SDN.

Controlador SDN	Características importantes	Lenguaje de programación	Versión	Año de creación
OpenDaylight	Código abierto y modular	Java	Beryllium	2013
ONOS	Escalabilidad y rendimiento	Java	2.0	2014
Floodlight	Implementación en Java	Java	12	2011
Ryu	Ligero y fácil de usar	Python	4.34	2011
2NOX	Enfocado en investigaciones académicas	C++	0.5.0	2008
Beacon	Diseño simple y extensible	Java	1.0	2011
Juniper Contrail	Enfocado en redes virtuales y nubes	Python	5.1.3	2013

Nota. En esta tabla se muestra algunos aspectos sobre los controladores SDN como su lenguaje de programación, año y la característica principal que lo compone.

2.3 VXLAN

El protocolo de encapsulación conocido como Virtual Extensible LAN (VXLAN) ha sido propuesto para implementar una red superpuesta dentro de una infraestructura ya existente en la Capa 3. Una red superpuesta se configura como una red virtual construida sobre las tecnologías preexistentes de Capa 2 y Capa 3, con el propósito de respaldar arquitecturas

informáticas flexibles. VXLAN proporciona a los profesionales de redes la capacidad de expandir entornos de cómputo en la nube, al mismo tiempo que garantiza un aislamiento lógico para las aplicaciones y los inquilinos en la nube (Ruano et al., 2017).

Virtual Extensible LAN (VXLAN) es una tecnología de virtualización de redes creada para solucionar los problemas de escalabilidad que se presentan al utilizar la tecnología VLAN convencional en grandes implementaciones de computación en la nube. Inicialmente desarrollada por VMware y Cisco, VXLAN surgió como una solución para abordar las dificultades identificadas en dichos entornos. Otros defensores clave de esta tecnología actualmente incluyen a Juniper Networks, Arista Networks, Broadcom, Citrix y Red Hat (Nadeau & Gray, 2013).

Los componentes claves de la tecnología VXLAN dentro de la red son:

- ***VTEP (Punto de Extremo de Túnel Virtual):*** Es un componente que se encarga de establecer el túnel VXLAN y gestiona los procesos de encapsulación y desencapsulación del paquete.
- ***VNI (Identificador de Red Virtual):*** Es un identificador de 24 bits que permite hasta 16 millones de redes lógicas diferentes, por medio, de un valor único para cada red lógica.
- ***Túnel VXLAN:*** Es el túnel lógico que transporta los paquetes Ethernet encapsulados dentro de paquetes UDP sobre la red IP.
- ***Red Underlay:*** Red física sobre la que se construye la red, también conocida como Red Subyacente.
- ***Red Overlay:*** Red virtual construida encima de la red física, con la capacidad de crear topologías lógicas sin importar la red física, recibe el nombre de Red Superpuesta.
- ***SPINE:*** Es el dispositivo o dispositivos que interconectan los equipos VTEPs y permiten la comunicación en la red VXLAN.

2.3.1 Direccionamiento de VXLAN

VXLAN utiliza Dispositivos de Punto de Túnel de Extensión de Virtualización (VTEP) para la ejecución de la encapsulación y desencapsulación del paquete VXLAN. Cada VTEP dispone de dos interfaces: la primera conectada al segmento LAN local para respaldar la comunicación de los puntos finales locales a través de un puente, y la segunda interfaz destinada a la red IP de transporte (Naranjo E & Salazar G, 2017).

La interfaz que se conecta a la red de transporte cuenta con una dirección IP única que sirve para identificar el dispositivo VTEP, también referida como infraestructura VLAN. El VTEP utiliza esta dirección IP para encapsular tramas Ethernet y luego transmite los paquetes encapsulados a la red de transporte a través de dicha interfaz (Naranjo E & Salazar G, 2017).

A través de esta interfaz, el VTEP no solamente descubre otros VTEP en segmentos VXLAN remotos, sino que también obtiene información sobre direcciones MAC remotas asociadas a asignaciones VXLAN. Este proceso es referido como "ARPing" debido al uso del protocolo ARP según las especificaciones del estándar IEEE 802.3 (Naranjo E & Salazar G, 2017).

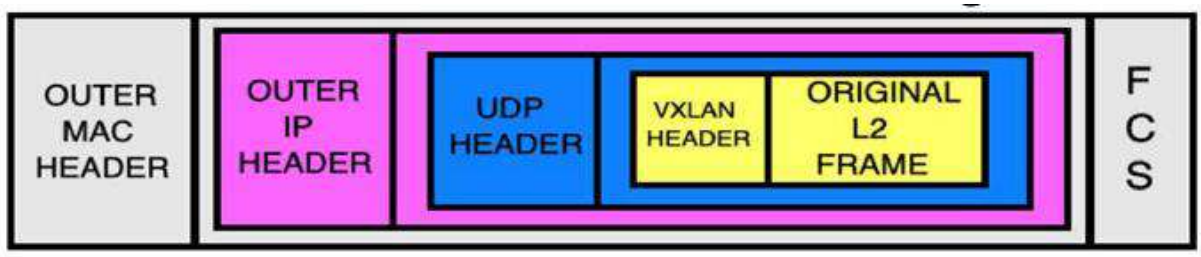
2.3.2 Formato de paquete VxLAN

VXLAN utiliza una técnica de encapsulación similar a VLAN para empaquetar tramas Ethernet de Capa 2, que se basan en direcciones MAC, dentro de paquetes UDP de Capa 3. Con la encapsulación MAC-in-UDP, VxLAN crea una abstracción de Capa 2 para las máquinas virtuales (VM), sin importar su ubicación. Esta técnica se asemeja al propósito para el que se desarrolló LISP, cuyo término dentro del marco de VxLAN, se refiere al protocolo de Separación de Localizador/Identificación, que aborda desafíos específicos relacionados con la gestión de direcciones en entornos de red, asignando un identificador único a cada dispositivo y utiliza direcciones IP solo para la localización.

Existen 4 encabezados para el formato del paquete utilizado, como se observa en la **Figura 4**.

Figura 4

Formato de Paquete VXLAN.



Nota. Recuperado de (Dutt et al., 2014)

- **VXLAN Header**

Según (Dutt et al., 2014), la cabecera VXLAN posee una longitud de 8 bytes, la cual, está compuesta por 4 campos que son:

- **Flags:** el campo consiste en 8 bits que muestran la secuencia "R-R-R-R-I-R-R-R". La presencia de "1" en el flag denominado "I" indica la autenticidad de un VNI (Número de Identificación de Vehículo). Los demás bits, identificados como "R", deben tener un valor de "0" al transmitir la trama y no se consideran durante la recepción.
- **VXLAN Network Identifier (VNI o VNID):** es un campo de 24 bits usado para identificar cada red "Overlay". Esto permite crear más de 16 millones de segmentos VXLAN dentro del mismo dominio administrativo. Cada VXLAN tiene un VNI único, y cada máquina que forma parte de un VXLAN se identifica mediante su dirección MAC y su VNI. Esto hace posible que haya máquinas con direcciones MAC repetidas en diferentes redes VXLAN.
- **Campos reservados:** Existen dos campos reservados en la cabecera VXLAN. Uno de ellos está situado entre el campo de flags y el campo VNI y el otro se encuentra situado

al final de la cabecera VXLAN. Estos campos tienen una longitud de 24 y 8 bits respectivamente, ambos deben ser enviados con valor "0" y son ignorados en la recepción del paquete.

- **UDP Header**

El encabezado VXLAN y la trama Ethernet original se emplean como información UDP. En el encabezado UDP, el puerto de destino (correspondiente al puerto VXLAN) se establece en 4789, mientras que el puerto de origen (puerto de origen UDP) se calcula utilizando un algoritmo hash basado en la trama Ethernet original (Huawei, 2022).

- **Outer IP Header**

Se trata del encapsulamiento del encabezado IP externo. En este encabezado, la dirección IP de origen (IP externa de origen) corresponde a la dirección IP del VTEP vinculado a la máquina virtual de origen, mientras que la dirección IP de destino (IP externa de destino) corresponde a la dirección IP del VTEP conectado a la máquina virtual de destino (Huawei, 2022).

- **Outer MAC Header**

Se refiere al encapsulamiento del encabezado Ethernet externo. En este encabezado, la dirección MAC de origen (MAC de origen) se corresponde con la dirección MAC del VTEP conectado a la máquina virtual de origen, mientras que la dirección MAC de destino (MAC de destino) se relaciona con la dirección MAC del siguiente salto a lo largo de la ruta hacia el VTEP de destino (Huawei, 2022.

De igual manera, en las redes VXLAN, el plano de control y el plano de datos son dos componentes separados que desempeñan funciones diferentes.

- ✓ **Plano de datos**

En VXLAN para el plano de datos, es importante contar con una infraestructura de red subyacente para llevar a cabo el reenvío requerido. Esta infraestructura es fundamental, puesto que, posibilita la comunicación unidireccional entre los dispositivos finales conectados a la VXLAN. Al mismo tiempo, esta infraestructura de red puede utilizarse para direccionar el tráfico multidestino hacia los dispositivos finales conectados a un dominio de transmisión L2 compartido en la red superpuesta. Este tipo de tráfico, conocido como BUM, comprende la difusión, la unidifusión desconocida y la multidifusión (Naranjo E & Salazar G, 2017).

- ✓ **Plano de control**

El plano de control forma el método que posibilita la disponibilidad y el aprendizaje en el ámbito de VXLAN. La base de su funcionamiento es lo que se conoce como el comportamiento de inundación y aprendizaje. En términos más simples, la inundación y el aprendizaje representan el proceso mediante el cual un VTEP, al no tener información acerca de la ubicación de una dirección MAC concreta, procede a enviar tramas a grupos de multidifusión asociados con VXLAN. La aplicación de la multidifusión se utiliza con el propósito de ofrecer una gestión más eficaz del tráfico multidestino (Naranjo E & Salazar G, 2017).

2.4 Mininet

Mininet es en esencia un emulador de código abierto utilizado como plataforma de pruebas para redes, siendo reconocido como una herramienta de importancia en la investigación de redes definidas por software (SDN) con OpenFlow, puesto que, utiliza hosts virtuales, switches y enlaces. Mininet posee la capacidad de construir redes en un solo núcleo del sistema operativo. Los hosts virtuales generados por Mininet permiten la ejecución de aplicaciones de red basadas en Unix/Linux, lo que facilita la experimentación y el desarrollo en este ámbito (Mininet Project Contributors, 2022).

2.4.1 Características de Mininet

De acuerdo con (Mininet Project Contributors, 2022), Mininet establece características que lo hacen un emulador con ciertas mejoras. A continuación, se listan las principales:

- Simplifica la cooperación entre varios desarrolladores, permitiendo trabajar de manera independiente en una topología compartida.
- Facilita realizar pruebas sistemáticas de regresión, las cuales pueden repetirse con facilidad y son sencillas de empaquetar.
- Integra una Interfaz de Línea de Comandos (CLI) que permite el reconocimiento de la configuración de la topología y OpenFlow, lo que hace más fácil depurar y ejecutar pruebas en toda la red.
- Facilita la creación de topologías exactas y suministra un conjunto necesario de estructuras topológicas con parámetros que pueden ser modificados.
- Ofrece una interfaz de programación en Python que es de fácil uso y es adaptable para la creación y experimentar con redes.

3 CAPITULO III. ESCENARIO PARA IMPLEMENTACIÓN

Este capítulo presenta las herramientas que generan el entorno de pruebas que nos brindará el apoyo para la ejecución del sistema basado en SDN y VxLAN.

3.1 Requerimientos para instalación

Mininet, como en un emulador en gran medida versátil, facilita el diseño y la emulación de redes SDN sin la necesidad de recurrir a los procesos de equipos de hardware y grandes infraestructuras físicas. Pues, ofrece un entorno controlable y eficiente. También, permite la fácil integración de controladores SDN, lo que simplifica la gestión de redes mediante el protocolo OpenFlow. Además, soporta las configuraciones más avanzadas, como VXLAN, convirtiéndola en una herramienta fuerte para probar y validar soluciones para cuestiones de redes virtualizadas.

La **Tabla 2** indica los requisitos mínimos necesarios para instalar la herramienta Mininet: un procesador de al menos 2 GHz, RAM (4 GB recomendados), 10 GB de espacio en disco y comúnmente es suficiente contar con una sola tarjeta de red en la mayoría de los casos. Sin embargo, si se pretende conectar la red simulada a redes externas o realizar pruebas avanzadas, se recomienda poseer al menos dos interfaces de red.

Tabla 2

Requisitos para instalación de MININET.

Requisito	Descripción
CPU	Procesador de al menos 2 GHz o superior
RAM	Mínimo: 2 Gb, recomendado: 4Gb o superior
Almacenamiento	Mínimo: 10 Gb de espacio libre en disco
Tarjeta de red	Múltiples interfaces Ethernet para conectar nodos

Fuente: (Mininet Project Contributors, 2022)

Por otra parte, el controlador SDN Floodlight proporciona un equilibrio destacado en varios aspectos clave para proyectos de redes, donde se puede evidenciar la escalabilidad, simplicidad y flexibilidad. La capacidad de operación con bajos recursos, junto con su soporte para API RESTful, lo convierten en una herramienta ideal para la gestión integrada de SDN y VXLAN. Esta adaptabilidad hace que sea especialmente adecuado para ajustarse a las necesidades en cuestión de la infraestructura, brindando una solución eficiente y robusta para la administración de redes avanzadas.

La **Tabla 3** detalla los requisitos necesarios para instalar el controlador SDN Floodlight, los cuales son: sistema operativo Ubuntu 16 LTS, JDK 8 o superior (recomendándose OpenJDK u Oracle JDK), y la versión más reciente de Maven. Cumplir estos requisitos asegura una instalación correcta y el mejor rendimiento del controlador para gestionar redes SDN y VXLAN.

Tabla 3

Requerimiento para instalación de controlador SDN Floodlight.

Requisito	Descripción
Sistema Operativo	Ubuntu 16 LTS
Java	JDK (Java Development Kit) 8 o superior Se recomienda OpenJDK o Oracle JDK
Maven	Versión más reciente

Fuente: (Open Networking Foundation, 2024)

Para cerrar, el sistema operativo que cumple con los atributos necesarios es Linux, tomando en cuenta versiones como Ubuntu 16.04, ya que ofrece compatibilidad comprobada al trabajar tanto con Mininet como con el controlador Floodlight debido a las bibliotecas que maneja. Esta versión de Ubuntu permite facilitar la creación de un entorno funcional estable,

debido a su soporte extendido y robusto, garantizando tanto todas las bibliotecas como dependencias necesarias para evitar problemas. Esto es importante, ya que en versiones más recientes del sistema operativo se generan conflictos de compatibilidad que dificultan el funcionamiento correcto de estas herramientas.

La **Tabla 4** muestra los requisitos del sistema operativo, recomendando Ubuntu 16 LTS junto con la versión más reciente de VirtualBox, ya sea actualizada o recomendada. Estos elementos garantizando la compatibilidad y estabilidad necesarias para el funcionamiento de las herramientas.

Tabla 4

Requerimientos para instalación de Sistema Operativo.

Requisito	Descripción
Sistema Operativo	Ubuntu 16 LTS
Virtual Box	Versión recomendada o actualizada

Fuente: (Mininet Project Contributors, 2022)

3.2 Escenario propuesto

A continuación, se presenta la topología de red propuesta para la interconexión de VTEPs y hosts utilizando VXLAN (Virtual Extensible LAN) y OpenFlow, bajo una arquitectura de red definida por software (SDN), gestionada por el controlador Floodlight. Esta configuración permite una mayor escalabilidad y flexibilidad en la red, garantizando la separación lógica de segmentos de red a través de túneles VXLAN y el control centralizado del tráfico mediante el uso de OpenFlow. Los componentes clave incluyen los switches SDN, VTEPs, hosts y el controlador Floodlight, que en conjunto proporcionan una infraestructura

optimizada para la gestión eficiente del tráfico y la conectividad entre los diferentes segmentos de la red.

La idea principal es mostrar cómo la integración de estas tecnologías puede mejorar la creación de redes virtuales, destacando sus aplicaciones y beneficios en múltiples contextos. Este proyecto no solo tiene como fin optimizar la planificación y diseño inicial de la redes, sino también proporcionar una evaluación útil como base para futuras implementaciones de redes virtuales en variedad de situaciones.

La iniciativa es proporcionar soluciones prácticas y efectivas que ayuden a las personas a comprender y adoptar mejor estas tecnologías emergentes. Se espera que esto ayude en el desarrollo de redes virtuales más adaptables y escalables que pueden ajustarse a las demandas del entorno tecnológico actual.

Teniendo en cuenta lo nombrado, se establecen los siguientes componentes para la construcción del escenario, siguiendo las siguientes indicaciones para el establecimiento de IP's y VNI dentro de la topología, esto debido a que en una arquitectura de red que implementa tecnologías de Redes Definidas por Software (SDN) y Redes de Área Local Extensible Virtual (VXLAN), la asignación de direcciones IP es esencial en la segmentación lógica de la red y en el aislamiento de dominios de difusión.

Esta segmentación se realiza de manera que se minimicen conflictos entre subredes, se maximice la eficiencia en la administración de direcciones y se garantice una estructura de red escalable y segura. A continuación, se detalla la lógica aplicada de todos los componentes para la topología presentada:

1. **Controlador SDN Floodlight**:

En la topología, el controlador SDN (para este caso, el controlador Floodlight) la IP se configurada como 127.0.0.1 para pruebas locales, indicando así que el controlador se ejecuta

en el mismo host donde se simula la red. Sin embargo, en un entorno de producción, el controlador debería ubicarse en una red privada dedicada, como una subred 10.0.x.x o 192.168.x.x, para asegurar el acceso seguro y controlado. Esta subred debe estar separada de las subredes de los hosts, permitiendo un canal de gestión independiente para evitar interferencias en el tráfico de datos o algún otro tipo de inconveniente.

Además, otro punto que se debe conocer es que el controlador se encuentra conectado al puerto 6653 (puerto defecto del controlador Floodlight).

2. **Puntos finales del túnel VXLAN (VTEPs)**:

Dentro de la topología se establecen tres VTEPs para el escenario que incluyen de manera específica los aspectos siguientes:

- **VTEP 1:** Identificado con la dirección IP 192.168.0.2, se encuentra conectado a cuatro hosts con direcciones IP en la subred 10.0.0.1 – 10.0.0.4.
- **VTEP 2:** Identificado con la dirección IP 192.168.10.2, se encuentra conectado a cuatro hosts con direcciones IP en la subred 10.0.0.5 – 10.0.0.8.
- **VTEP 3:** Identificado con la dirección IP 192.168.100.2, se encuentra conectado a cuatro hosts con direcciones IP en la subred 10.0.0.9 – 10.0.0.12

- **Asignación IP para los VTEP**

La función de cada switch VTEP VXLAN (VTEP 1, VTEP 2 y VTEP 3) es la de ser el responsable de encapsular y desencapsular el tráfico de las redes virtuales, permitiendo la extensión de dominios de capa 2 sobre una infraestructura de capa 3. Es así como, para evitar conflictos, cada VTEP VXLAN se coloca en una subred distinta (192.168.0.2 para VTEP 1, 192.168.10.2 para VTEP 2, y 192.168.100.2 para VTEP 3). Este enfoque facilita la

administración y control de las distintas instancias de VXLAN, al mismo tiempo que permite una identificación clara de cada VTEP dentro de la infraestructura de red.

Estos rangos IP pueden seleccionarse dentro de cualquier espacio de direcciones privadas, como el rango 10.0.0.0/8, 172.16.0.0/12, o 192.168.0.0/16. Esto facilita la separación del tráfico y evita que haya conflictos con direcciones IP públicas.

- **Asignación de IP para hosts**

Los hosts conectados a cada VTEP VXLAN se configuran dentro de la subred 10.0.0.0/24, pero se asignan diferentes bloques de direcciones IP en función del VTEP al que están conectados:

  - Los hosts en VTEP 1 utilizan las direcciones 10.0.0.1 – 10.0.0.4.
  - Los hosts en VTEP 2 utilizan las direcciones 10.0.0.5 - 10.0.0.8.
  - Los hosts en VTEP 3 utilizan las direcciones 10.0.0.9 - 10.0.0.12.

La estructura de VXLAN permite reutilizar el mismo espacio de direcciones (10.0.0.x) en diferentes dominios de difusión, simulando una red de capa 2 distribuida sobre una infraestructura de capa 3. Esto proporciona la flexibilidad de asignar los mismos rangos IP en múltiples ubicaciones sin generar conflictos, ya que el identificador VXLAN (VNI) diferencia cada dominio de difusión.

Esta configuración también facilita la migración y escalabilidad de los servicios, permitiendo la extensión de la red sin tener que reasignar direcciones IP, ya que cada segmento está claramente definido a través del VNI en lugar de depender únicamente de las subredes de capa 3.

3. Asignación de VNI

El VNI (VXLAN Network Identifier) es un identificador de red en VXLAN que permite segmentar el tráfico de diferentes redes virtuales dentro de una infraestructura compartida. Cada VNI actúa de manera similar a una VLAN, pero en un entorno de red superpuesto (overlay network), donde los VNIs permiten la creación de múltiples dominios de difusión dentro de una misma infraestructura física de red.

Los VNIs deben ser únicos en la red para evitar conflictos y garantizar que el tráfico de cada red virtual se mantenga aislado. En la mayoría de las implementaciones, se utiliza un número de 24 bits para el VNI, lo que permite tener hasta 16 millones de VNIs diferentes (de 1 a 16,777,215), mucho más que el límite de 4096 IDs en las VLAN tradicionales.

La asignación de VNIs se planifica de acuerdo con las necesidades de segmentación de la red. Cada red virtual o segmento lógico que se quiera crear tendrá su propio VNI, además, el VNI puede ser asignado arbitrariamente o según un esquema de asignación definido en la red.}

El VNI elegido para cada segmento debe ser consistente en todos los dispositivos de la red que participan en el tráfico VXLAN. Esto significa que cualquier switch, router o controlador que gestione tráfico para la VXLAN0, debe reconocer que el VNI 200 pertenece a este segmento.

En redes más grandes, los VNIs se pueden asignar según zonas, departamentos o roles específicos dentro de la organización, lo que facilita el control y la gestión del tráfico.

4. OFPort

En la topología, el funcionamiento de los puertos en los switches OpenFlow es un elemento clave para la comunicación dentro de la red definida por software (SDN). Los puertos de OpenFlow, comúnmente conocidos como OFPorts, plasman las interfaces físicas o virtuales que los switches utilizan para conectar dispositivos (hosts) o establecer túneles, como en el caso

de VXLAN. Dichos puertos son asignados automáticamente por el software que implementa el mismo protocolo OpenFlow, como Open vSwitch (OVS)[3], dependiendo de la topología definida y las conexiones creadas en el entorno de simulación.

A continuación, la **Tabla 5** presenta el direccionamiento IPv4 que ayudará a solventar la convergencia del escenario propuesto.

Tabla 5

Tabla de direccionamiento del escenario en IPv4.

Dispositivo	Interfaz	Dirección IP	Máscara
Controlador Floodlight	Eth0	127.0.0.1	255.0.0.0
VTEP 1	-------	192.168.0.2	**255.255.255.0**
VTEP 2	-------	192.168.10.2	**255.255.255.0**
VTEP 3	-------	192.168.100.2	**255.255.255.0**
h1	Eth0	10.0.0.1	**255.255.255.0**
h2	Eth0	10.0.0.2	**255.255.255.0**
h3	Eth0	10.0.0.3	**255.255.255.0**
h4	Eth0	10.0.0.4	**255.255.255.0**
h5	Eth0	10.0.0.5	**255.255.255.0**
h6	Eth0	10.0.0.6	**255.255.255.0**
h7	Eth0	10.0.0.7	**255.255.255.0**
h8	Eth0	10.0.0.8	**255.255.255.0**
h9	Eth0	10.0.0.9	**255.255.255.0**

[3] Open vSwitch (OVS): Switch virtual de código abierto diseñado para redes definidas por software (SDN) y virtualización de redes.

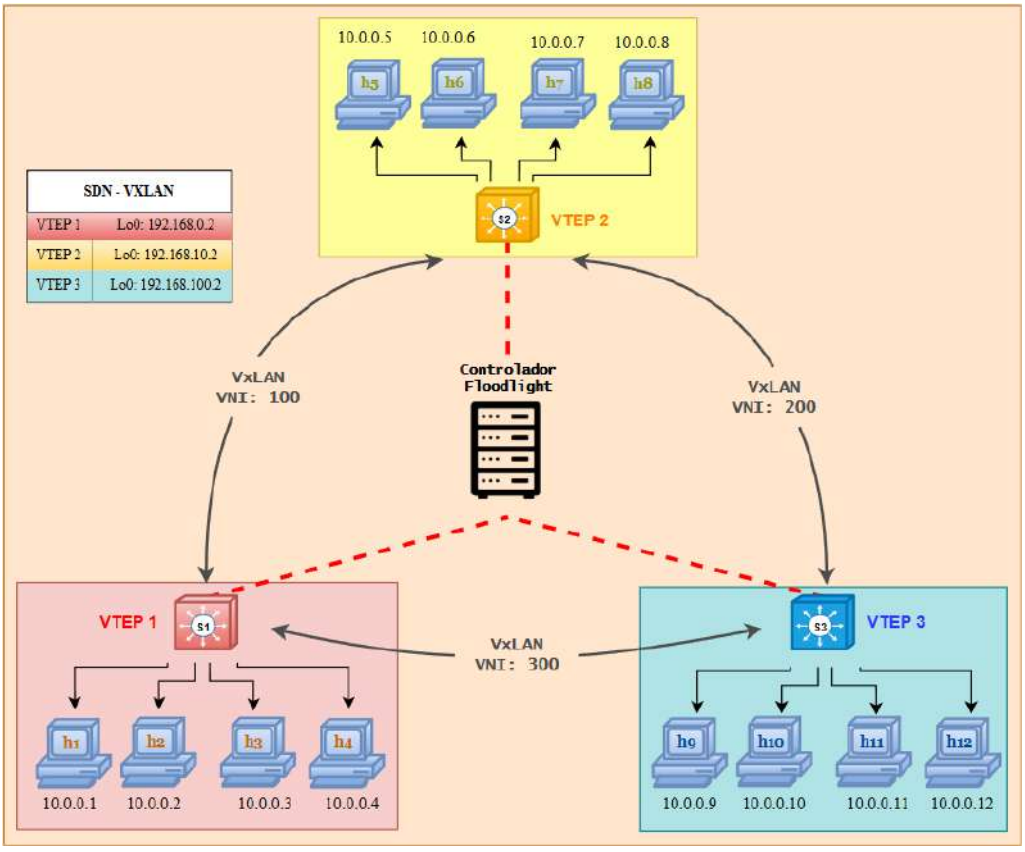
h10	Eth0	10.0.0.10	**255.255.255.0**
h11	Eth0	10.0.0.11	**255.255.255.0**
h12	Eth0	10.0.0.12	**255.255.255.0**

Tras los conceptos anteriormente explicados, se realiza un par de escenarios que permiten el caso de estudio planteado, es así como, se demuestra lo siguiente:

- **ESCENARIO 1**

Figura 5

Escenario estructural base (SDN – VXLAN).



Nota: La figura muestra una infraestructura de red basada en VXLAN, gestionada a través de un controlador SDN Floodlight. La topología de la red está compuesta por tres dominios VXLAN, cada uno identificado por un VNI (Identificador de Red Virtual), sin necesidad de configuraciones adicionales.

- **ESCENARIO 2**

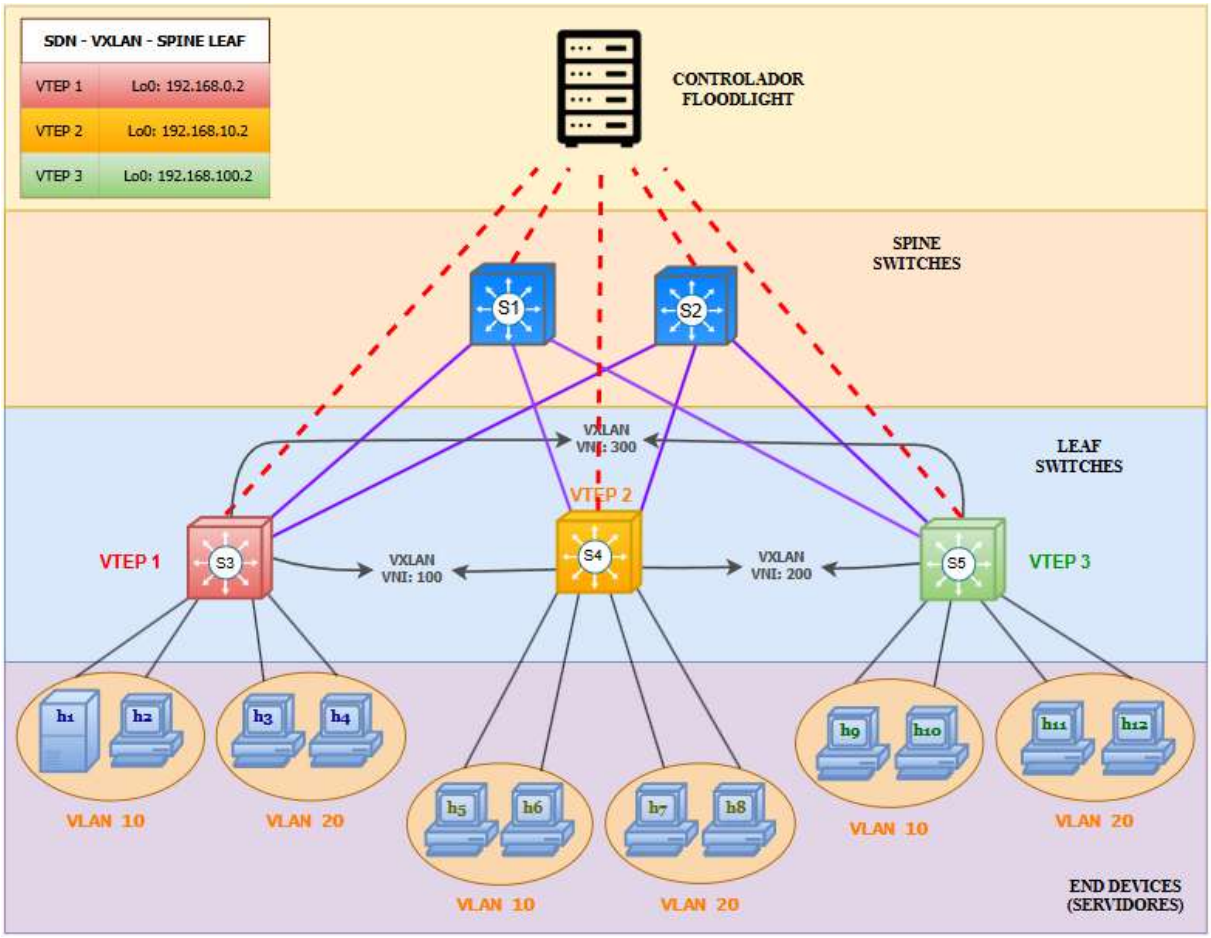
En este escenario, se adapta la implementación de un diseño de red como la arquitectura Spine-Leaf debido a la necesidad de mejorar las particularidades de una red como la escalabilidad, redundancia y eficiencia en la red SDN-VXLAN. En la arquitectura de la Figura 5, la interconexión entre VTEPs está limitada, lo que podía generar cuellos de botella y dificultades en la expansión de la red. Con Spine-Leaf, cada switch Leaf se conecta a múltiples switches Spine, lo que permite una distribución más equitativa del tráfico y un crecimiento flexible sin afectar el rendimiento de la infraestructura en general.

Otros factores clave incluyen la reducción de latencia y el balanceo de carga. El diseño de la red Spine-Leaf ofrece múltiples rutas entre los dispositivos, optimizando el flujo de tráfico mediante Equal-Cost Multipath (ECMP) y evitando la sobrecarga de enlaces específicos. Además, mejora la tolerancia a fallos, ya que la falla de un switch Spine no afecta la conectividad general, puesto que el tráfico logra redirigirse automáticamente por medio de otros enlaces activos, asegurando mayor estabilidad y continuidad operativa.

Desde la perspectiva VXLAN y SDN, esta topología con nueva estructura optimiza la gestión de túneles VXLAN, asegurando una mejorada distribución del tráfico encapsulado y brindando una interconexión entre VNIs sin generar congestión. También, el controlador SDN Floodlight ayuda a la gestión de manera eficiente de los flujos de red, permitiendo un control más dinámico y adaptable. Por estos argumentos, se ha decidido adaptar a la topología la arquitectura de red Spine-Leaf, a causa de que garantiza un entorno de pruebas más realista, representativo con relación a las redes modernas de alto rendimiento y con una administración centralizada más efectiva, como se representa en la **Figura 6**.

Figura 6

Escenario propuesto (SDN – VxLAN – Spine Leaf).



Conectividad y Propósito:

- La topología está diseñada para lograr la interconexión de múltiples VTEPs y sus respectivos hosts a través de VXLAN, asegurando la comunicación entre ellos mediante túneles sobre una red subyacente.

- El controlador Floodlight se encarga de gestionar la configuración de OpenFlow y la administración de los túneles VXLAN, controlando el tráfico entre los hosts a través de las distintas subredes y VTEPs.

- Esta infraestructura fue pensada en base a un escenario de simulación de una red de centros de datos distribuidos o como un simple escenario de pruebas de escalabilidad de redes SDN (Software-Defined Networking) y VXLAN con arquitectura Spine-Leaf.

3.3 Levantamiento de Topología sobre Python

Tras definir el diseño de la topología de red e instalar las herramientas pertinentes y configuradas de manera correcta, el paso siguiente es implementar el código Python que permite desplegar la red simulada. Para ello, se desarrolló un archivo con extensión `.py`, el cual establece una red red en Mininet utilizando Open vSwitch (OVS) y un controlador SDN remoto Floodlight. En primer lugar, se define un controlador remoto con la dirección IP y puerto 6653, el cual gestionará la red mediante el protocolo OpenFlow. Posteriormente, se agregan los switches de la capa Spine, que actuarán como el núcleo de la red, y los switches de la capa Leaf, que funcionarán como VTEPs (VXLAN Tunnel Endpoints), cada uno asociado a un VNI (100, 200 y 300) para segmentar el tráfico en diferentes dominios VXLAN.

A continuación, en la Figura 7, el código establece la interconexión entre los switches Spine y Leaf, creando enlaces que aseguran la comunicación dentro de la red. Se definen los hosts y su asignación a VLANs específicas, distribuyéndolos en VLAN 10 y VLAN 20. Después, se crean los hosts y se asocian con los switches Leaf correspondientes, garantizando que cada grupo de dispositivos esté correctamente conectado a su VTEP respectivo. Finalmente, se generan los enlaces entre cada host y su switch Leaf, permitiendo que los dispositivos finales se comuniquen a través de la infraestructura VXLAN definida en la topología. Esta configuración es clave para simular una red SDN-VXLAN basada en la arquitectura Spine-Leaf dentro del entorno de Mininet.

Figura 7

Construcción de la Topología en Mininet por medio del script desarrollado en Python.

```
# Agregar el controlador remoto Floodlight, especificando su IP y puerto de escucha OpenFlow
controller = net.addController('c0', ip='127.0.0.1', port=6653)

# Agregar switches Spine (núcleo de la red)
s1 = net.addSwitch('s1', cls=OVSSwitch)  # Switch Spine 1
s2 = net.addSwitch('s2', cls=OVSSwitch)  # Switch Spine 2

# Agregar switches Leaf, que actuarán como VTEPs en la red VXLAN
vtep1 = net.addSwitch('s3', cls=OVSSwitch)  # VTEP 1 asociado al VNI 100
vtep2 = net.addSwitch('s4', cls=OVSSwitch)  # VTEP 2 asociado al VNI 200
vtep3 = net.addSwitch('s5', cls=OVSSwitch)  # VTEP 3 asociado al VNI 300

# Conectar los switches Spine con los switches Leaf (VTEPs)
for spine in [s1, s2]:  # Iterar sobre cada switch Spine
    for leaf in [vtep1, vtep2, vtep3]:  # Iterar sobre cada switch Leaf
        net.addLink(spine, leaf)  # Crear enlace entre el Spine y el Leaf

# Definir los hosts y asignarlos a VLANs específicas
vlan_10_hosts = ['h1', 'h2', 'h5', 'h6', 'h9', 'h10']  # Hosts en VLAN 10
vlan_20_hosts = ['h3', 'h4', 'h7', 'h8', 'h11', 'h12']  # Hosts en VLAN 20

# Crear hosts y asignarlos a cada VTEP correspondiente
hosts_vtep1 = [net.addHost(h) for h in ['h1', 'h2', 'h3', 'h4']]  # Hosts conectados a VTEP 1
hosts_vtep2 = [net.addHost(h) for h in ['h5', 'h6', 'h7', 'h8']]  # Hosts conectados a VTEP 2
hosts_vtep3 = [net.addHost(h) for h in ['h9', 'h10', 'h11', 'h12']]  # Hosts conectados a VTEP 3

# Conectar los hosts a los switches Leaf correspondientes (VTEPs)
for h in hosts_vtep1:
    net.addLink(h, vtep1)  # Conectar cada host al VTEP 1
for h in hosts_vtep2:
    net.addLink(h, vtep2)  # Conectar cada host al VTEP 2
for h in hosts_vtep3:
    net.addLink(h, vtep3)  # Conectar cada host al VTEP 3
```

Después, se configuran las interfaces VXLAN en los switches dentro de la topología de la red. Para esto, se define una lista llamada vxlan_config, que contiene varias tuplas, donde cada una representa la configuración de un túnel VXLAN en un switch específico. Cada entrada en la lista indica el switch (VTEP) donde se configurará la interfaz VXLAN, el nombre de la interfaz, la dirección IP local del VTEP, la dirección IP remota del otro VTEP con el que se establecerá el túnel y el VNI (VXLAN Network Identifier) que se usará para segmentar el tráfico de red.

En la configuración, el switch s3 (VTEP 1) establece túneles VXLAN con los switches s4 (VTEP 2) y s5 (VTEP 3), utilizando los VNIs 100 y 300, respectivamente. De manera similar, el switch s4 crea túneles VXLAN con s3 y s5, manejando los VNIs 100 y 200. Por último, el switch s5 establece túneles con s4 y s3, asegurando la conectividad de los VNIs 200 y 300. Esta configuración permite que los dispositivos en diferentes segmentos de la red puedan

comunicarse de manera eficiente mediante la encapsulación VXLAN, independientemente de la ubicación física de los hosts.

El propósito de la configuración mostrada en la Figura 8 es permitir la interconexión de redes virtuales a través de una infraestructura SDN-VXLAN, asegurando que el tráfico se encapsule y se transporte correctamente entre los diferentes dominios VXLAN. Esto facilita la segmentación del tráfico y la escalabilidad de la red, manteniendo la compatibilidad con la arquitectura Spine-Leaf sobre la que está basada la topología.

Figura 8

Configuración de túneles VXLAN.

```
# Configurar interfaces VXLAN en los switches correctamente
vxlan_config = [
    ('s3', 'vxlan301', '192.168.0.2', '192.168.10.2', 100),
    ('s3', 'vxlan302', '192.168.0.2', '192.168.100.2', 300),
    ('s4', 'vxlan401', '192.168.10.2', '192.168.0.2', 100),
    ('s4', 'vxlan402', '192.168.10.2', '192.168.100.2', 200),
    ('s5', 'vxlan501', '192.168.100.2', '192.168.10.2', 200),
    ('s5', 'vxlan502', '192.168.100.2', '192.168.0.2', 300)
]
```

inalmente, otro aspecto importante es la definición de las reglas OpenFlow en los VTEPs para gestionar el tráfico VXLAN en la red. Se establece un bucle que recorre los tres VTEPs (vtep1, vtep2, vtep3) y sus respectivos VNIs (100, 200 y 300), asignando reglas específicas para manejar la encapsulación y desencapsulación del tráfico VXLAN en Open vSwitch (OVS).

La primera regla agrega un flujo que asocia una VLAN entrante con un VNI específico. Si un paquete llega con una etiqueta VLAN particular (dl_vlan), se modifica su campo tun_id con el valor del VNI correspondiente y luego se reenvía usando output:NORMAL. Esto permite que el tráfico VLAN se encapsule dentro de un túnel VXLAN y se envíe correctamente a su destino.

La segunda regla maneja el tráfico VXLAN a través del protocolo UDP en el puerto 4789, que es el puerto estándar para VXLAN. Esta regla permite el procesamiento normal de paquetes que llegan a este puerto, asegurando que el switch los maneje correctamente sin bloquearlos.

Por último, la tercera regla se activa cuando un paquete llega con un VNI específico (tun_id). En este caso, la regla elimina la etiqueta VLAN (strip_vlan) y reenvía el paquete a través de output:NORMAL, permitiendo que el tráfico llegue a su destino final sin la encapsulación de VXLAN. Este mecanismo es clave para que los paquetes VXLAN sean correctamente interpretados y entregados a los hosts en la red.

En conjunto, estas reglas presentadas en a **Figura 9** permiten la interoperabilidad entre VLANs y VXLANs dentro de la red definida por software, asegurando que los paquetes sean correctamente encapsulados, transportados y desencapsulados en los diferentes segmentos de la infraestructura.

Figura 9

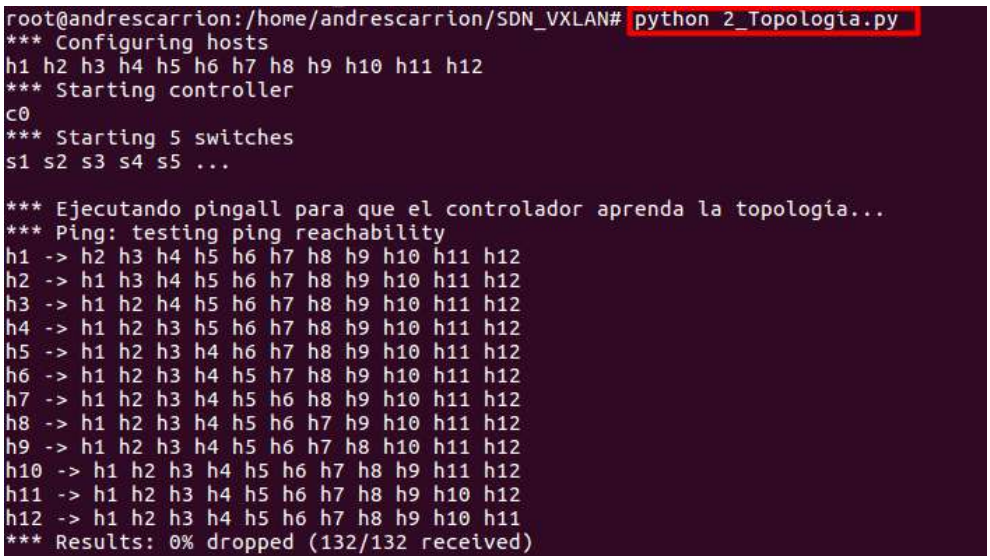
Reglas agregadas para Openflow.

```
# Agregar reglas OpenFlow para manejar VXLAN correctamente
openflow_rules = [
    ('s3', 10, 100), ('s3', 20, 100), ('s3', 10, 300), ('s3', 20, 300),
    ('s5', 10, 200), ('s5', 20, 200)
]
for switch, vlan, vni in openflow_rules:
    net.get(switch).cmd(
        'ovs-ofctl add-flow {} "priority=300,dl_vlan={},actions=set_field:{}->tun_id,output:NORMAL"'.format(switch, vlan, vni))
    net.get(switch).cmd('ovs-ofctl add-flow {} "priority=320,udp,tp_dst=4789,actions=NORMAL"'.format(switch))
    net.get(switch).cmd(
        'ovs-ofctl add-flow {} "priority=250,tun_id={},actions=strip_vlan,output:NORMAL"'.format(switch, vni))
```

Una vez terminada las configuraciones necesarias dentro del script para crear la topología pertinente aplicamos el comando para verificar la ejecución de este con ayuda del comando PYTHON y el nombre del script como se muestra en la **Figura 10.**

Figura 10

Ejecución de script con las configuraciones de red SDN con soporte VXLAN.



La **Figura 11** muestra el estado del controlador Floodlight, una herramienta utilizada en redes definidas por software (SDN) para gestionar y controlar interruptores OpenFlow. En la parte superior, se puede ver el "Hostname", que en este caso es localhost:6633, indicando la dirección y el puerto donde se está ejecutando el controlador.

También se confirma que el controlador está funcionando correctamente, con el atributo "Healthy" marcado como verdadero. Además, se muestra el "Uptime", que indica el tiempo que el controlador ha estado operativo, en este caso, 30 segundos desde su inicio.

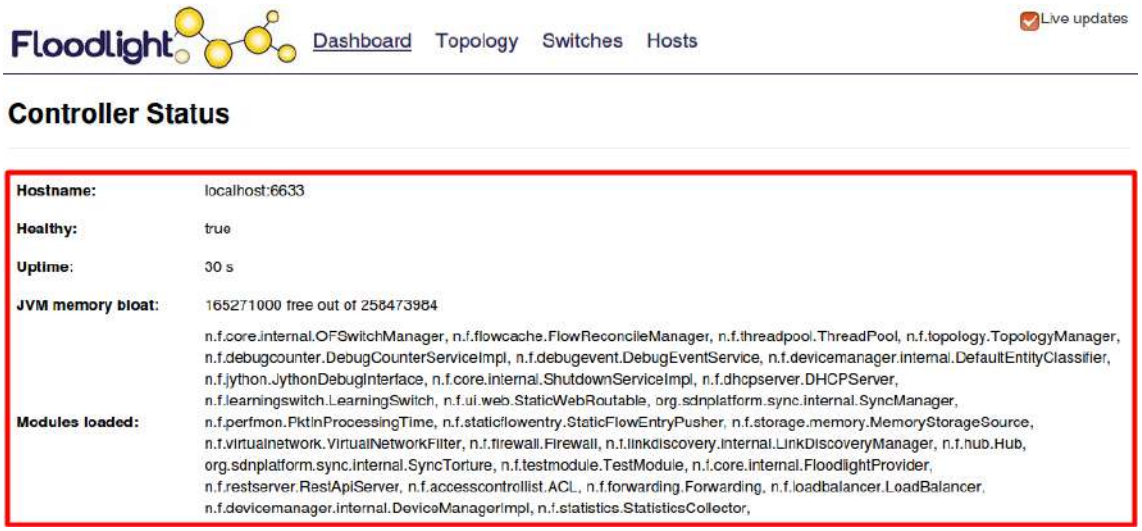
Otro dato relevante es el uso de memoria de la máquina virtual Java (JVM), indicado en "JVM memory bloat". Aquí se muestra la memoria utilizada y disponible en el sistema, en este caso, 16,527,100 bytes libres de un total de 25,847,394 bytes asignados.

Finalmente, se detalla la lista de módulos cargados en el controlador, bajo "Modules loaded". Estos módulos incluyen funcionalidades como la gestión de flujos (Flow Reconcile Manager), descubrimiento de enlaces (Link Discovery Manager), servidor DHCP, balanceo de carga (Load Balancer) y recopilación de estadísticas (Statistics Collector), entre otros. Estos

componentes permiten que el controlador administre la red de forma eficiente y ofrezca diversas funcionalidades avanzadas en el entorno SDN.

Figura 11

Estado del controlador Floodlight y módulos cargados.



Por otro lado, en las siguientes figuras (**Figura 12 y Figura 13**) se presenta la topología de red gestionada por el controlador Floodlight, una herramienta clave en redes definidas por software (SDN). Esta visualización centralizada, accesible a través de la web, permite monitorear la conectividad entre switches y hosts en tiempo real, ofreciendo información esencial como las direcciones MAC, IP y los puertos asociados a cada dispositivo. Floodlight desempeña un papel crucial como gestor de la infraestructura de red, ayudando a los administradores a identificar rápidamente el estado de los enlaces y posibles problemas de conectividad.

Figura 12

Dashboard del panel de control del controlador Floodlight.

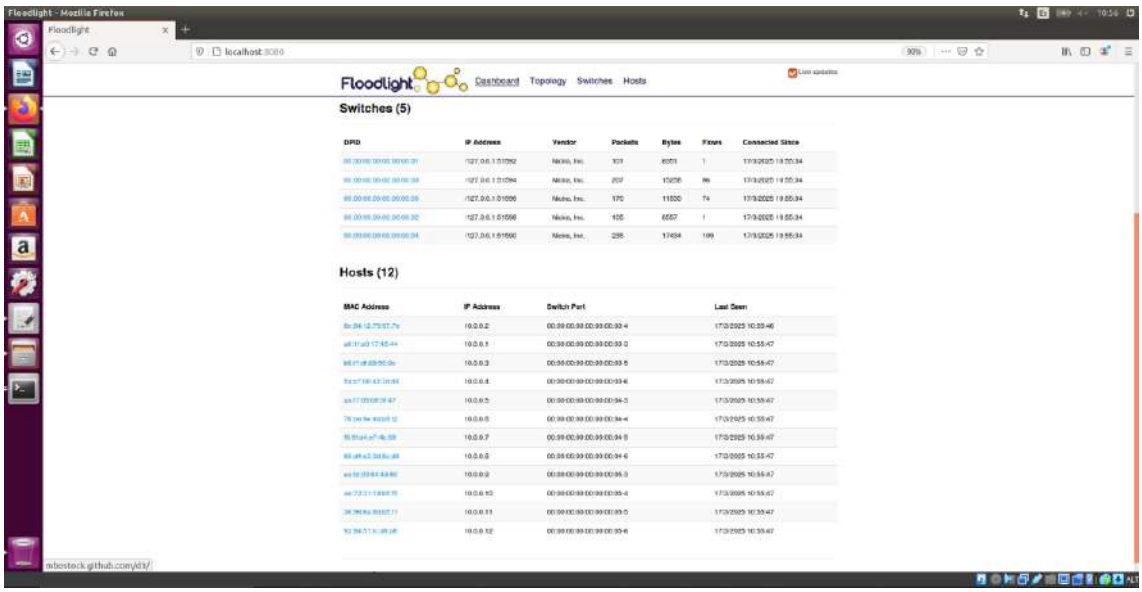
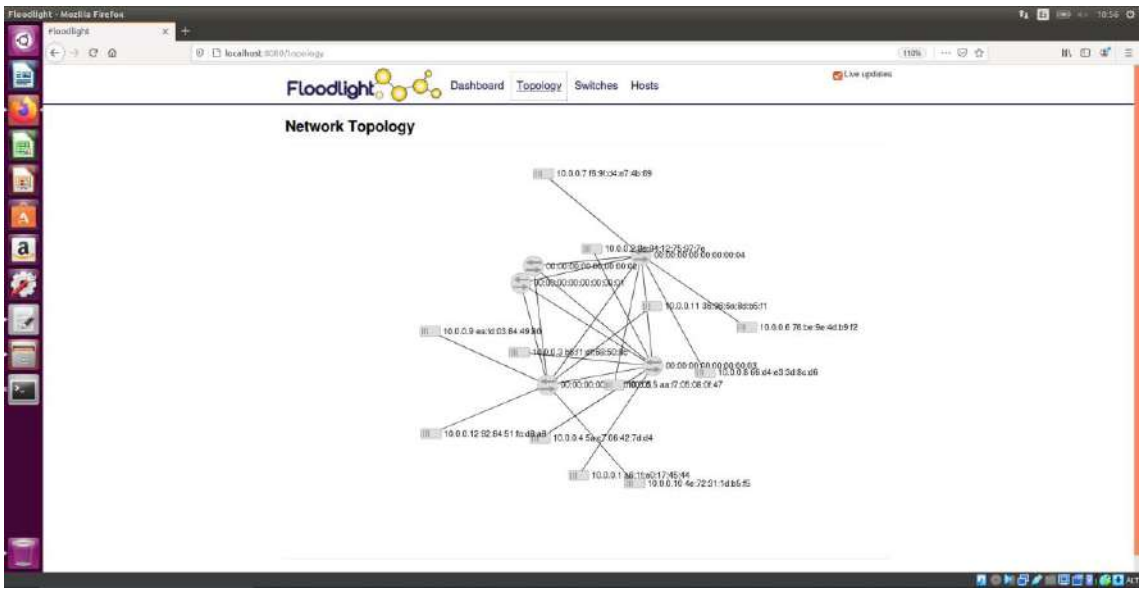


Figura 13

Topología de red visualizada en el controlador Floodlight.



3.4 Importancia del protocolo OpenFlow

En la topología SDN y VXLAN, OpenFlow actúa como el protocolo estándar para la comunicación entre el controlador SDN (Floodlight) y los switches de red. Una de las

principales funciones de OpenFlow al integrarse con un controlador SDN como Floodlight en una topología basada en VXLAN es que permite que el controlador programe y administre de forma centralizada las tablas de flujo de los switches SDN. Estas tablas contienen reglas específicas que determinan cómo el tráfico debe ser procesado y enrutado dentro de la red, lo que aporta flexibilidad y control total sobre la infraestructura.

En el contexto de redes con VXLAN, OpenFlow facilita el redireccionamiento del tráfico hacia los túneles virtuales mediante la configuración de reglas en los switches. Estas reglas permiten encapsular los paquetes en VXLAN y asignarlos al identificador de red virtual (VNI) correspondiente, asegurando que el tráfico llegue al destino correcto a través del túnel adecuado. Esto simplifica la comunicación entre diferentes servidores o dominios dentro de la red, como se observa en la topología implementada.

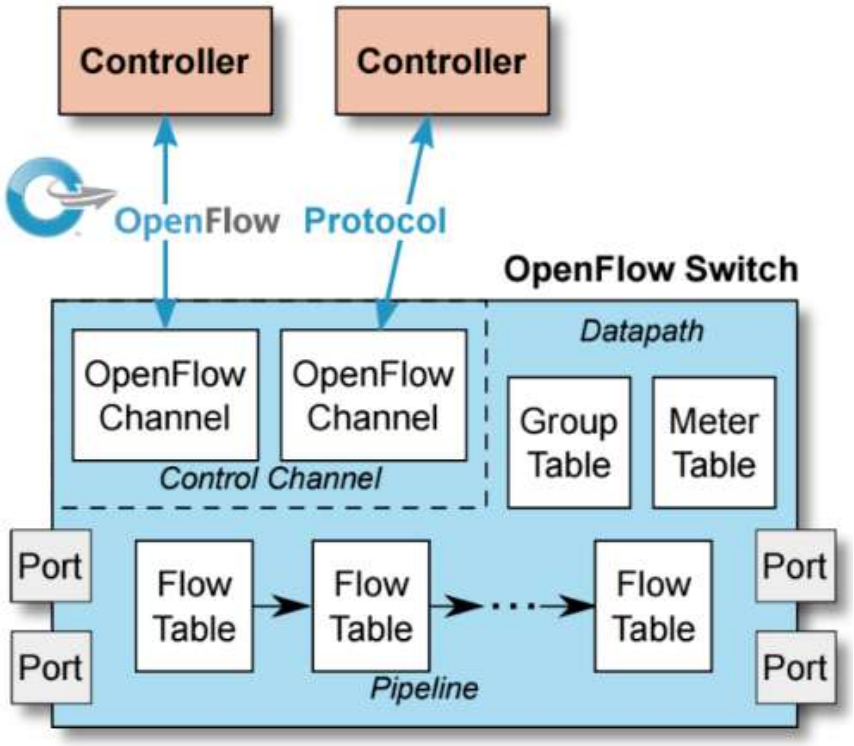
Además, OpenFlow permite aplicar políticas específicas sobre el tráfico, como el aislamiento entre VNIs. Esto garantiza que el tráfico de cada red virtual definida por VXLAN permanezca independiente, evitando interferencias entre redes lógicas que comparten la misma infraestructura física.

La capacidad de gestión y control centralizado que ofrece OpenFlow mejora significativamente la eficiencia, escalabilidad y seguridad de la red, convirtiendo a la combinación de SDN y VXLAN en una solución robusta para infraestructuras modernas y distribuidas.

A continuación, en la **Figura 14**, se presenta la arquitectura de un switch OpenFlow, destacando la separación entre el canal de control y el canal de datos. Se muestran las tablas de flujo, grupo y medidores, así como la interacción con el controlador a través del protocolo OpenFlow.

Figura 14

Componentes principales de un conmutador OpenFlow.



Fuente: (Acosta & Ortega, 2023)

El protocolo OpenFlow es un estándar clave en las redes definidas por software (SDN), que permite la separación del plano de control y el plano de datos. Esto se logra mediante la comunicación entre un controlador externo y los interruptores de red a través del protocolo OpenFlow. A continuación, se describen los componentes principales:

1. Conmutador OpenFlow

El Conmutador o Switch OpenFlow está compuesto por diversos elementos clave que permiten su funcionamiento eficiente dentro de una red definida por software. En primer lugar, se encuentran las tablas de flujo (Flow Tables), las cuales contienen entradas que incluyen campos de coincidencia, contadores y un conjunto de instrucciones. Los campos de coincidencia identifican paquetes basándose en criterios específicos, como direcciones IP o

puertos. Los contadores registran estadísticas relacionadas con el tráfico, como el número de paquetes y bytes procesados. Por su parte, el conjunto de instrucciones define las acciones a realizar cuando un paquete coincide con una entrada. La coincidencia inicia en la primera tabla y puede continuar a través de otras tablas adicionales en la tubería del conmutador (flujo lógico de procesamiento que los paquetes siguen dentro del switch a través de sus múltiples tables de flujo).

Otro componente importante es la tabla de grupos (Group Table), que permite implementar acciones avanzadas como el balanceo de carga o la replicación de paquetes. Esta tabla está organizada en grupos, cada uno con un identificador único y una lista de acciones asociadas. De manera similar, la tabla de medidores (Meter Table) se utiliza para medir el tráfico y aplicar políticas de control, como la limitación de ancho de banda. Los medidores se pueden configurar para controlar la velocidad del tráfico asociado a flujos específicos.

Finalmente, el conmutador cuenta con puertos, que son las interfaces a través de las cuales los paquetes se pueden reenviar. Estos pueden ser físicos, lógicos o reservados. Los puertos físicos corresponden a las interfaces de red físicas, mientras que los puertos lógicos son configurados por el conmutador, como los grupos de agregación de enlaces, túneles o interfaces de bucle invertido. Por otro lado, los puertos reservados están definidos específicamente por la especificación OpenFlow.

2. Canal de control OpenFlow (Control Channel)

El canal de control OpenFlow es un componente esencial en la arquitectura de las redes definidas por software (SDN), ya que establece la comunicación entre el controlador y el conmutador OpenFlow. A través de este canal, el controlador puede gestionar el conmutador mediante el intercambio de mensajes OpenFlow.

Estos mensajes permiten realizar diversas acciones clave, como agregar, actualizar y eliminar entradas en las tablas de flujo del conmutador, lo que asegura una gestión eficiente del tráfico en la red. Además, el controlador utiliza el canal de control para recibir estadísticas relacionadas con el tráfico y el estado operativo del conmutador, lo que facilita la supervisión y el análisis continuo del rendimiento de la red.

3. Controlador OpenFlow

El controlador OpenFlow es el componente central en la arquitectura de redes definidas por software (SDN). Su principal función es tomar decisiones sobre el enrutamiento y el reenvío del tráfico en la red, lo que lo convierte en una pieza clave para garantizar la eficiencia y el control de las comunicaciones.

Para llevar a cabo estas funciones, el controlador utiliza el canal de control OpenFlow, a través del cual puede configurar las tablas de flujo del conmutador, estableciendo las reglas que dictan cómo se procesan y enrutan los paquetes. Además, el controlador tiene la capacidad de monitorizar y gestionar la red de dos maneras principales: de forma reactiva, respondiendo a eventos o paquetes específicos, o de manera proactiva, anticipándose al tráfico esperado. Este enfoque dual permite una gestión dinámica y adaptable de la red, optimizando su rendimiento en diferentes escenarios.

4 CAPITULO IX: RESULTADOS

Este capítulo se centra en la recopilación y presentación de los resultados obtenidos durante el desarrollo del proyecto, utilizando el escenario propuesto para evaluar la integración de las tecnologías SDN y VXLAN, con el objetivo de ofrecer una visión clara y detallada de los logros alcanzados a lo largo del proceso. Asimismo, se realiza un análisis de estos resultados en relación con los objetivos planteadas en la investigación.

4.1 OpenFLow

En la topología diseñada, OpenFlow, como se mencionó anteriormente, juega un papel clave en la creación y gestión de los túneles VXLAN entre los dispositivos. El controlador Floodlight utiliza el protocolo OpenFlow para configurar las reglas de encapsulación en los switches conectados a las tres áreas propuestas en la red. Estas reglas determinan qué tráfico debe encapsularse para ser enviado a través de los túneles VXLAN y qué identificador de red virtual (VNI) debe asignarse a cada flujo de tráfico, asegurando que cada paquete se encapsule correctamente y llegue a su destino en la red virtual correspondiente.

Además, el controlador gestiona las rutas lógicas entre los VNIs, facilitando la interconexión eficiente entre los servidores. El controlador Floodlight asegura que los paquetes enviados entre las subredes virtuales (10.0.0.x) utilicen el túnel VXLAN adecuado para llegar a su destino, sin importar la ubicación física de las máquinas virtuales. Esto simplifica la administración de la red y asegura la conectividad entre los segmentos distribuidos.

Aunque no está demostrado explícitamente en la topología, el protocolo OpenFlow puede configurarse para implementar balanceo de carga y redundancia. Si existieran múltiples caminos físicos entre los servidores, Floodlight podría distribuir el tráfico de manera equitativa entre estos caminos, optimizando el uso de los recursos dispuestos en la red. Además, en caso de fallos en algún enlace, el controlador tiene la capacidad de redirigir el tráfico

automáticamente por rutas alternativas, asegurando la continuidad del servicio y mejorando la resiliencia de la infraestructura sin caer en pérdida totales.

Por último, OpenFlow sostiene el aislamiento lógico de los flujos de tráfico por medio de la definición de reglas que mantienen los paquetes de cada VNI separados entre sí. Por ejemplo, los flujos de tráfico asociados al VNI 200 (VXLAN 402 (VTEP 4) y VXLAN 501 (VTEP 5)) se gestionan de manera independiente de los flujos de los VNIs 100 y 300, garantizando que las diferentes redes virtuales que comparten la misma infraestructura física no interfieran entre sí, manteniendo la seguridad y la integridad del tráfico en toda la topología.

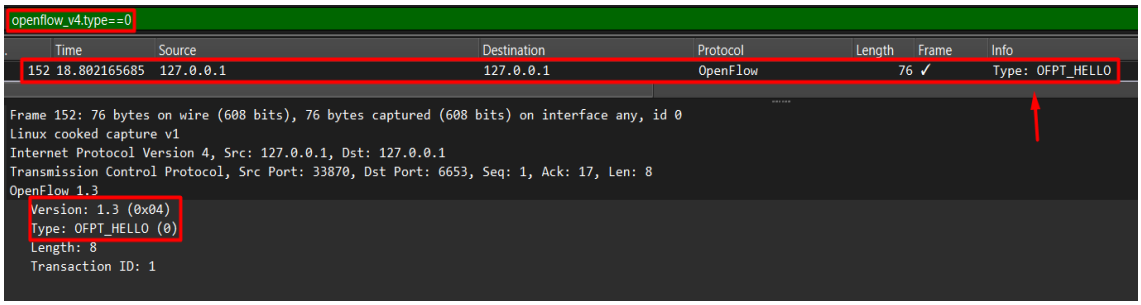
Ejemplo de paquetes típicos de OpenFlow

- **Hello**

El paquete **HELLO** de la **Figura 15**, es el primer mensaje intercambiado entre el controlador y el switch al establecer una conexión OpenFlow. Su propósito es negociar la versión del protocolo que ambos usarán. Este paquete es importante para garantizar que el controlador y el switch sean compatibles antes de cualquier otra interacción o intercambiar otros mensajes. En el analizador Wireshark, el identificativo es OFPT_HELLO con el tipo 0.

Figura 15

Paquete Hello.

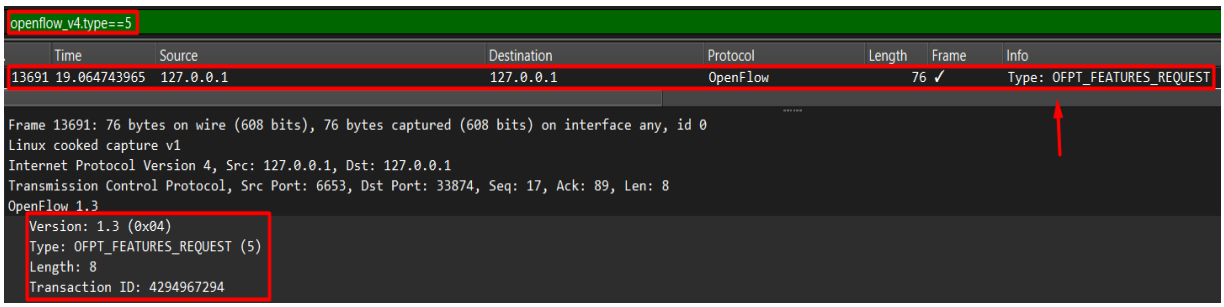


- **Features_Request y Features_Reply**

Estos mensajes se usan para que el controlador reciba la información sobre las capacidades del switch. El paquete **Features_Request** de la **Figura 16** es dirigido por el controlador solicitando información de las capacidades del equipo switch. Es usado para conocer ciertos detalles como: número de tablas, tamaño de los buffers y estadísticas que el switch soporta. Dicho mensaje es importante para que el controlador (Floodlight) ajuste su comportamiento según las cualidades del equipo switch. En el analizador Wireshark, se identifica el mensaje como OFPT_FEATURES_REQUEST con el tipo 5.

Figura 16

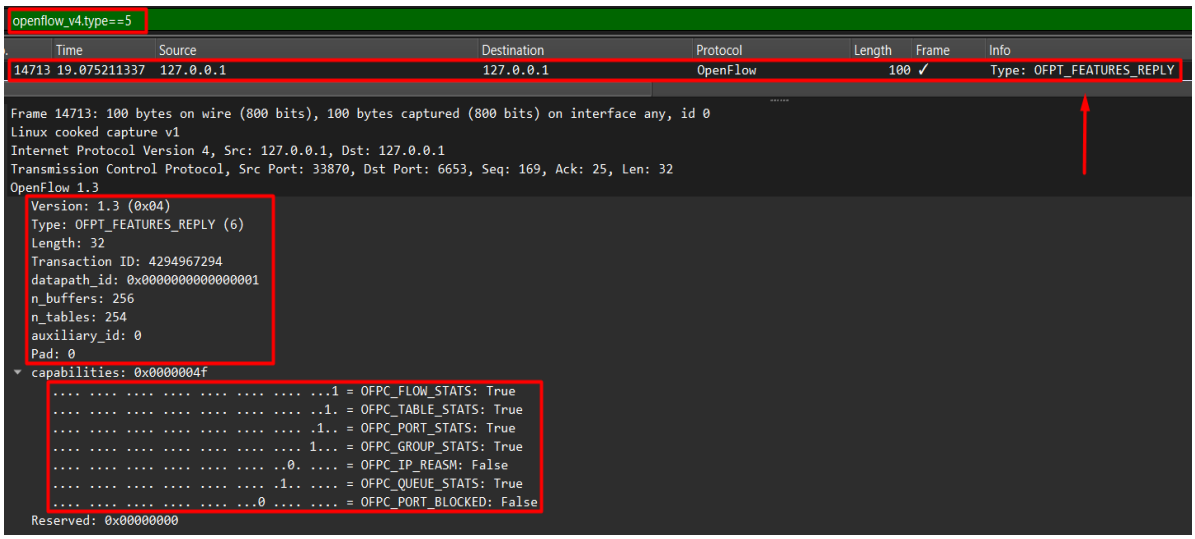
Paquete Features Request.



El paquete **Features_Reply** representado en la **Figura 17**, es la respuesta del switch al mensaje anterior **Features_Request**. Manaje la información detallada como: datapath_id (identificador único del switch), número de tablas, tamaño de los buffers, y capacidades soportadas (como estadísticas de puertos, tablas y flujos). Dicho mensaje facilita al controlador la planificación y gestión eficientemente de la red. En analizador Wireshark, se puede identificar como OFPT_FEATURES_REPLY con el tipo 6.

Figura 17

Paquete Features Reply.

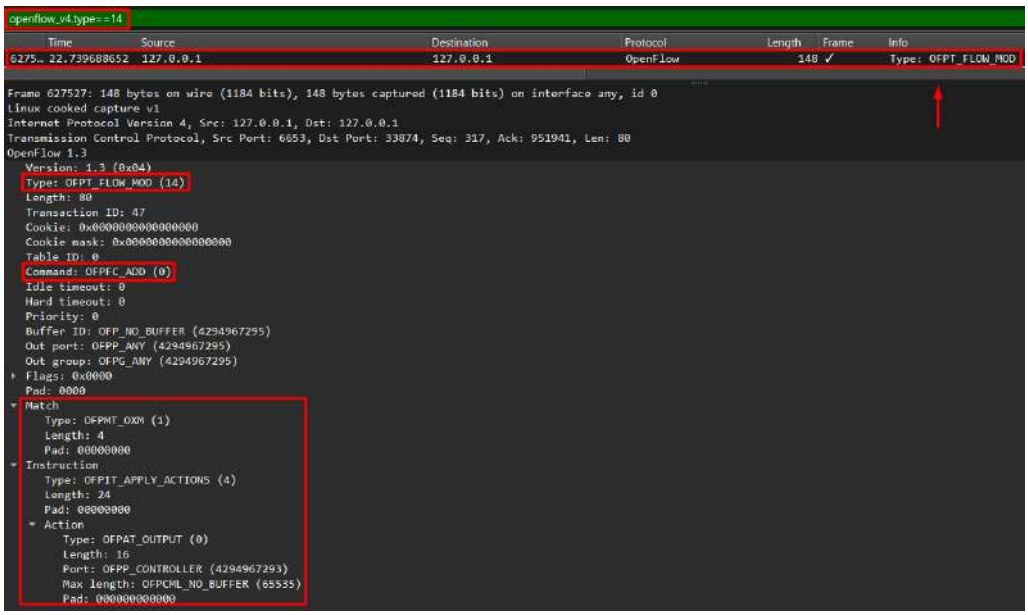


- **Flow_Mod**

El paquete **Flow_Mod** de la **Figura 18** es enviado por el controlador para agregar, modificar o eliminar reglas en las tablas de flujo del equipo switch. Es esencial para definir cómo los paquetes deben ser procesados por el switch. Maneja detalles tales como coincidencias (Match), instrucciones y acciones (Actions). Este mensaje es el núcleo de la funcionalidad de OpenFlow, ya que permite la programación dinámica del comportamiento del switch. En el analizador Wireshark, se logra identificar como OFPT_FLOW_MOD con el tipo 14.

Figura 18

Paquetes Packet Flow Mod.

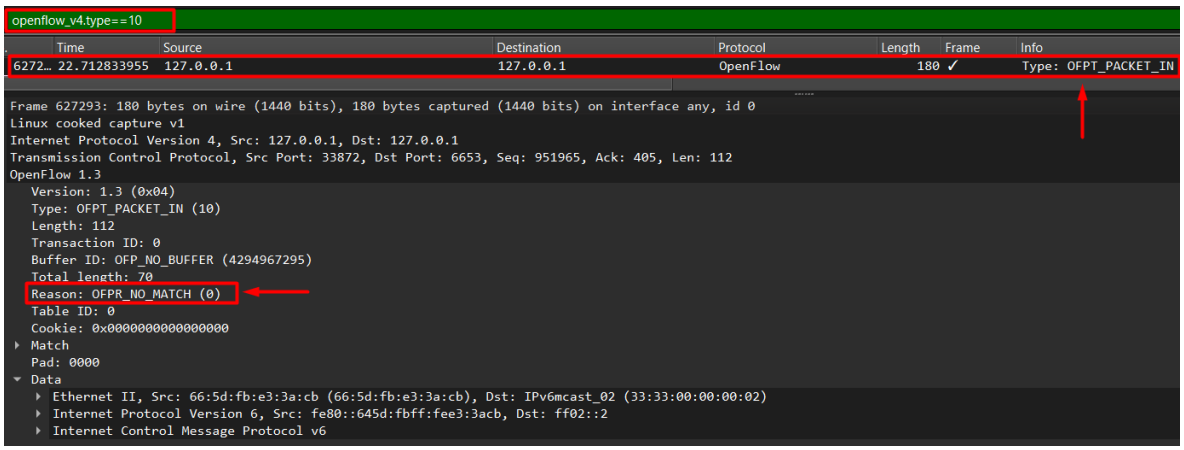


- **Packet_In y Packet_Out**

El mensaje **Packet_In** de la **Figura 19** es enviado por el switch al controlador cuando se produce la acción de que un paquete no coincide con alguna regla en las tablas de flujo o cuando está configurado para ser enviado al controlador. Maneja información del puerto de entrada y el paquete original. Dicho mensaje facilita al controlador tomar decisiones de cómo manejar paquetes desconocidos. En el analizador Wireshark, se puede identificar como OFPT_PACKET_IN con el tipo 10.

Figura 19

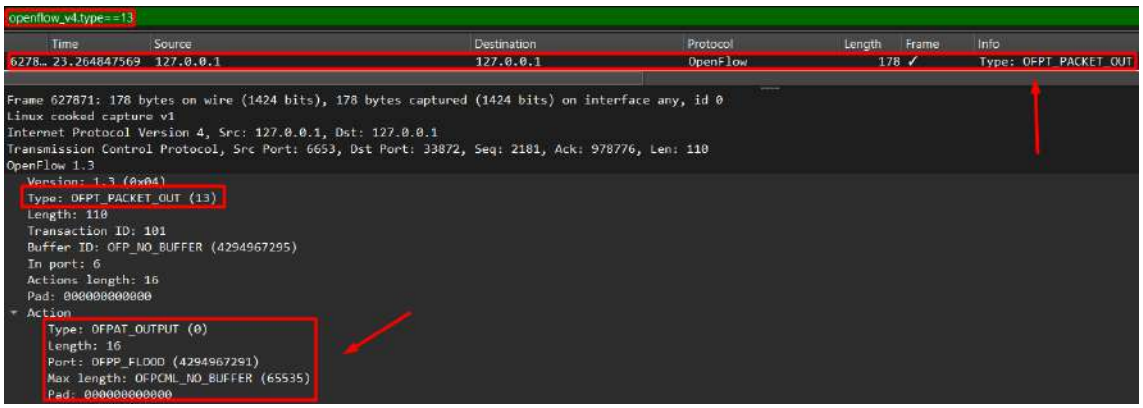
Paquetes Packet In.



El mensaje **Packet_Out** de la **Figura 20** es enviado por el controlador al switch para indicarle cómo manejar un paquete en concreto. Este mensaje, es el complemento de Packet_In, puesto que permite al controlador reenviar paquetes a un puerto específico o realizar acciones como flooding. Maneja instrucciones detalladas del destino y las acciones a realizar. En el analizador Wireshark, se puede identificar como OFPT_PACKET_OUT con el tipo 13

Figura 20

Paquetes Out.

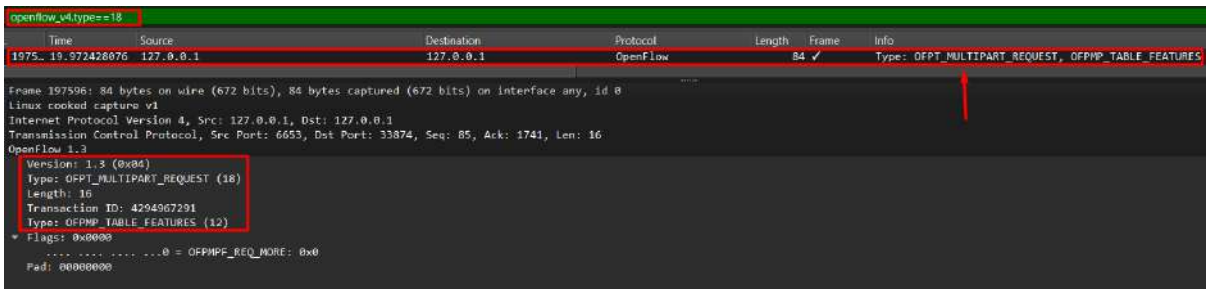


- **Multipart_Request y Multipart_Reply**

Estos mensajes se usan para la recopilación de estadísticas y detalles avanzados del tráfico, los flujos o los puertos en el equipo switch. El paquete **Multipart_Request** de la **Figura 21** es enviado por el controlador solicitando estadísticas o información detallada del switch, tales como estadísticas de flujos, tablas, puertos o grupos. Dicho mensaje es esencial para el monitoreo y análisis del estado de la red. En analizador Wireshark, se puede identificar como OFPT_MULTIPART_REQUEST con el tipo 18.

Figura 21

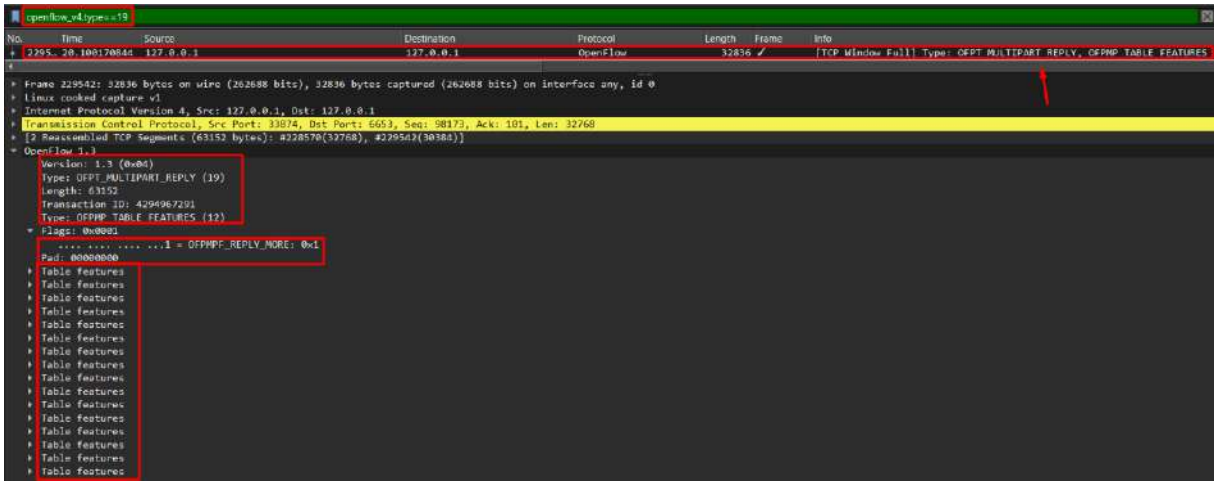
Paquetes Multipart Request.



El paquete **Multipart_Reply** de la **Figura 22** es la respuesta del switch al mensaje anterior Multipart_Request. Este maneja la información solicitada, como estadísticas detalladas o características concretas del switch. Dicho mensaje es importante para que el controlador tenga una visión actualizada del estado de la red. En el analizador Wireshark, se logra identificar como OFPT_MULTIPART_REPLY con el tipo 19.

Figura 22

Paquetes Multipart Reply.

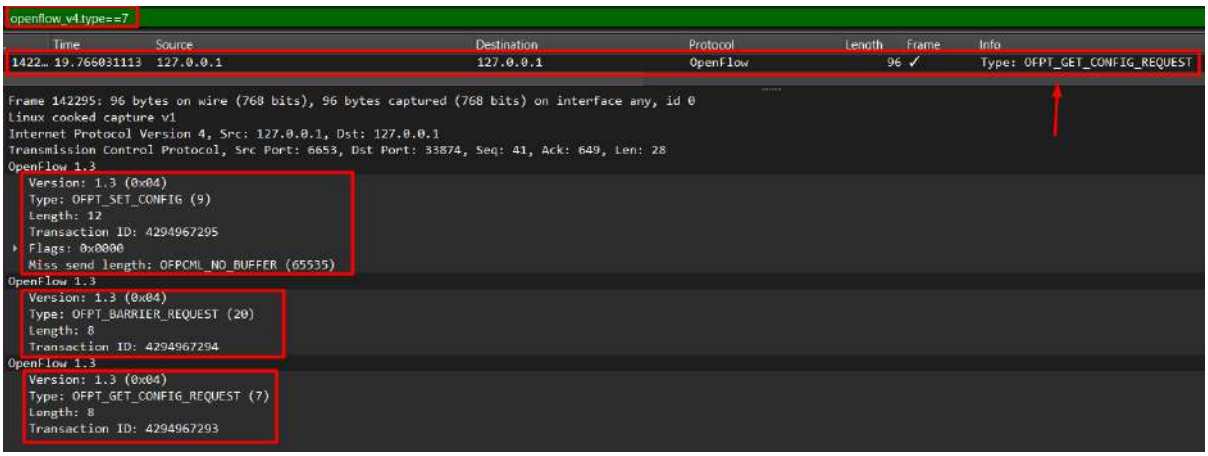


- **Paquetes Config Reply y Config Request**.

El paquete **Config_Request** de la **Figura 23** es el mensaje que envía el controlador hacia el switch. La idea idee es solicitar la actual configuración del dispositivo o establecer ciertos parámetros concretos, como, por ejemplo: cantidad de bytes de un paquete que no coinciden deben ser enviados al controlador. En el analizador Wireshark, se puede identificar como OFPT_GET_CONFIG_REQUEST con el tipo 7.

Figura 23

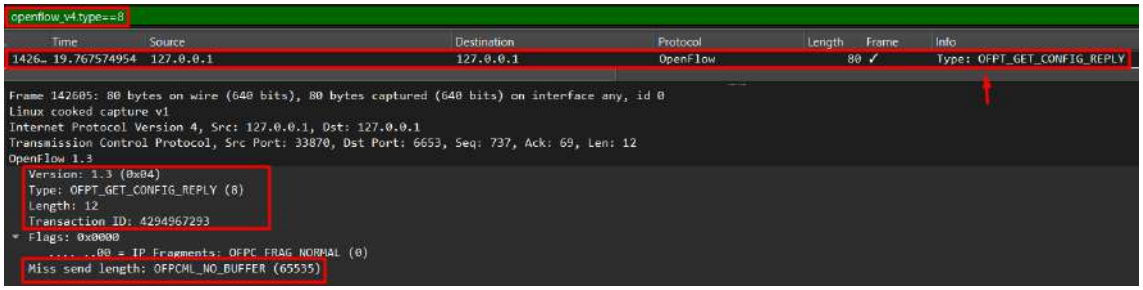
Paquetes Config Reply y Config Request.



El paquete **Config_Reply** de la **Figura 24** es la respuesta del switch al mensaje **Config_Request**. Maneja detalles de la configuración actual del equipo switch, como: comportamiento frente a paquetes no coincidentes (miss send length) y el manejo de fragmentos de paquetes. Dicho mensaje es de importancia para que el controlador comprenda cómo opera el switch y ajuste sus configuraciones de ser necesario. En el analizador Wireshark, se logra identificar como OFPT_GET_CONFIG_REPLY con el tipo 8.

Figura 24

Paquetes Config Reply.

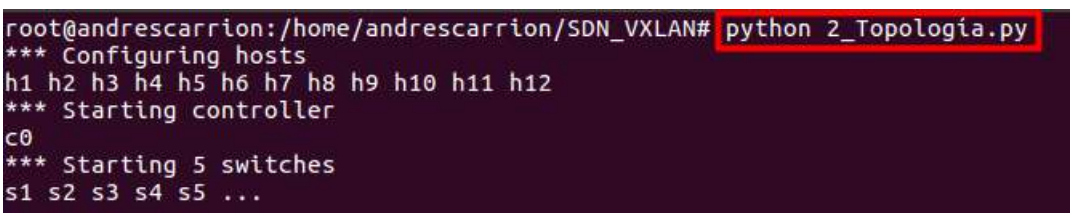


4.2 Pruebas de convergencia

Una vez establecido el escenario con todas las características programadas para su levantamiento se realiza la ejecución del entorno en un terminal del sistema operativo donde con la ayuda de la herramienta PYTHON anteriormente instalada iniciamos el script que contiene las configuraciones que nos permitirán el estudio del protocolo VxLAN con el uso de una arquitectura de red SDN. Para ello se hace uso del comando establecido en la **Figura 25**.

Figura 25

Ejecución de script para levantamiento de red.



Dentro del script para un arranque correcto se propone que ya una vez levantada cada una de las configuraciones se realice un reconocimiento del funcionamiento inmediato del escenario dentro de MININET con el comando PINGALL que hará la verificación de conectividad entre todos los hosts en la topología, generando una prueba de ping desde un host hacia los demás de la red uno a uno para comprobar si todos los equipos pueden comunicarse entre sí, como se muestra en la **Figura 26.**

Figura 26

Prueba de convergencia entre todos los hosts de la red implementada.

```
*** Ejecutando pingall para que el controlador aprenda la topología...
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
*** Starting CLI:
```

En SDN, el controlador no tiene conocimiento previo de la topología ni de los hosts conectados hasta que se genera tráfico. Realizar un ping fuerza la generación de tráfico entre hosts, lo que permite al controlador SDN aprender las direcciones IP, MAC y los puertos de conexión de los dispositivos conectados. Esto es esencial para que el controlador gestione el flujo de datos en la red y para que la información de los hosts se actualice y aparezca en la interfaz web del controlador. A continuación, se presentan algunos ejemplos de la comunicación entre los hosts de los distintos servidores mediante el comando ping en las **Figuras 27, 28 y 29**.

Figura 27

Prueba de ping del equipo h1 (vtep 1) a h5 (vtep 2).

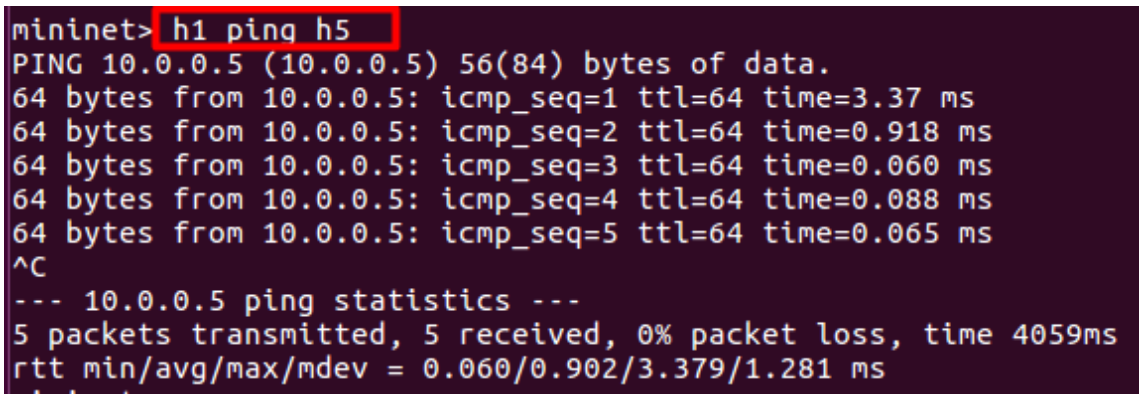


Figura 28

Prueba de ping del equipo h7 (vtep 2) a h12 (vtep 3).

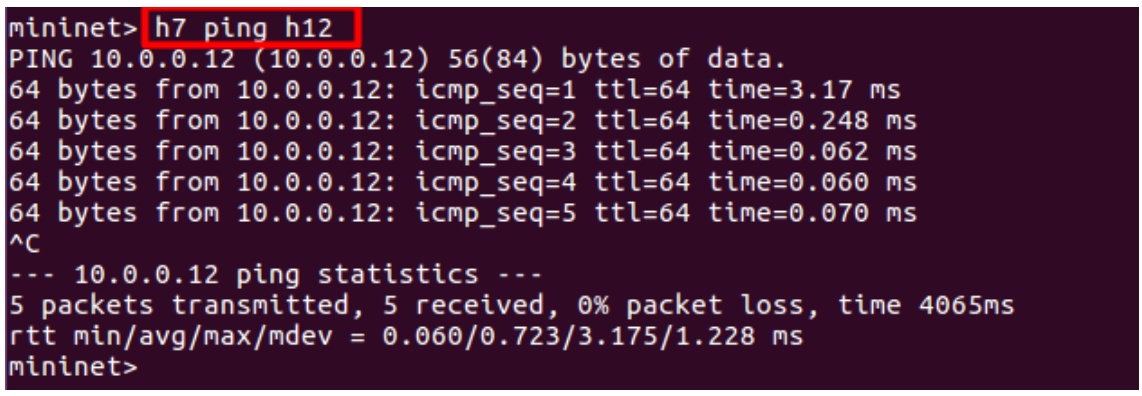
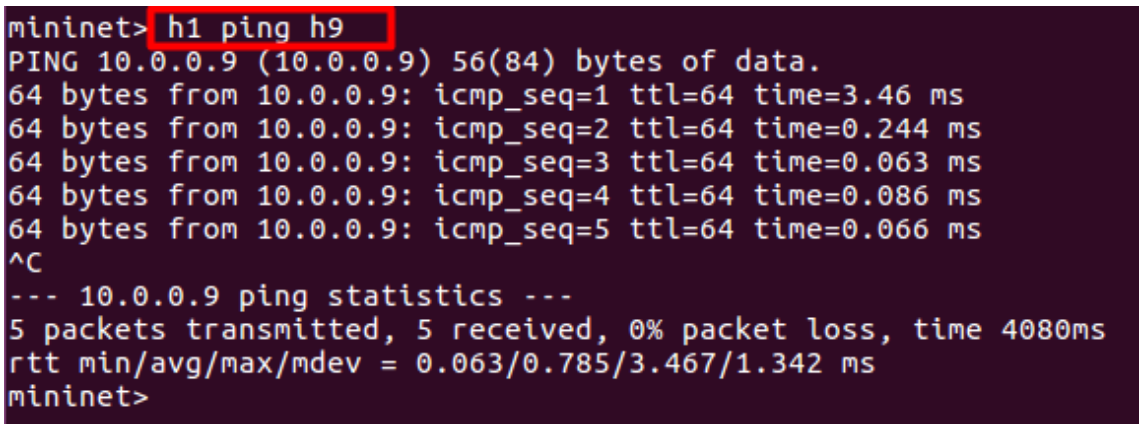


Figura 29

Prueba de ping del equipo h1 (vtep 1) a h9 (vtep 3).



4.2.1 Validación VXLAN con tráfico ICMP

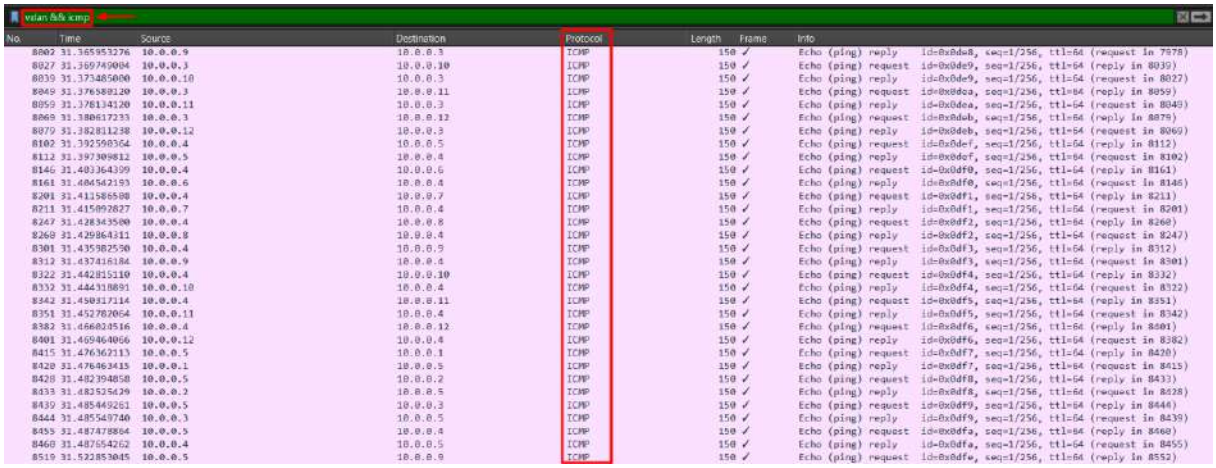
La **Figura 30** muestra el tráfico ICMP (Internet Control Message Protocol) generado por solicitudes y respuestas de **ping** entre los hosts de la red configurada con SDN y VXLAN.

Esta captura es clave para validar la conectividad entre los nodos dentro del dominio virtual, mostrando que el tráfico se enruta correctamente a través de los túneles VXLAN.

La conectividad exitosa entre los hosts en esta red virtualizada confirma el buen funcionamiento de los túneles VXLAN, permitiendo la comunicación entre nodos ubicados en diferentes segmentos físicos o lógicos, pero dentro del mismo dominio VXLAN. Además, demuestra que el controlador SDN gestiona eficientemente la topología de la red, controlando el flujo de datos y asegurando una entrega fiable de los paquetes ICMP.

Figura 30

Paquetes ICMP entre hosts de distintos VTEPs.

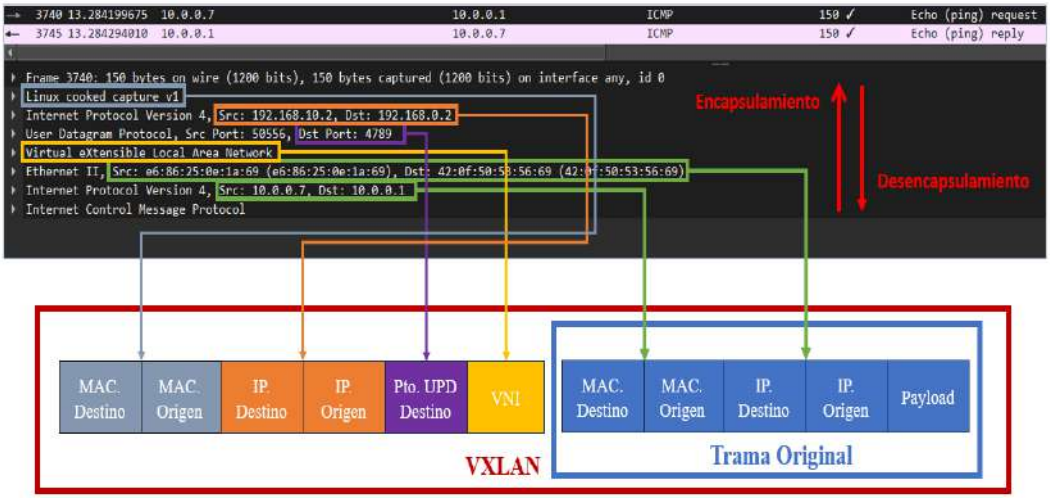


4.2.2 Encapsulamiento VXLAN dentro de la Red SDN.

La Figura 31 presenta todos los elementos que conforman un paquete VXLAN, evidenciando el proceso de encapsulación y desencapsulación, con el propósito de cumplir el formado definido por el protocolo, asegurando cada campo de la estructura según las especificaciones de VXLAN.

Figura 31

Encapsulamiento y desencapsulamiento del Paquete VXLAN.

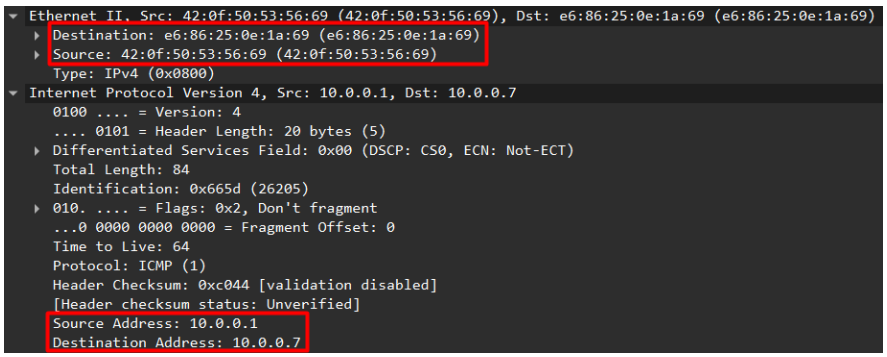


A continuación, se despliega el proceso de encapsulamiento que realiza el protocolo VXLAN a partir de un ejemplo visual obtenido del escenario implementado.

La **Figura 32**, indica el paquete que se envía o transmite entre los hosts ubicados dentro de la red virtual (Overlay). Bajo este contexto, los datos de relevancia son: el payload, IP de origen, IP de destino, MAC de origen y MAC de destino.

Figura 32

Original L2 Frame (Capa 3 interna).



La **Figura 33** proporciona información del funcionamiento del protocolo VXLAN (Virtual eXtensible Local Area Network), específicamente del encabezado del paquete

VXLAN, utilizado para la creación y gestión de redes virtualizadas a gran escala sobre infraestructuras IP (red superpuesta). En este paquete en concreto, el campo **Flags** (valor 0x0800) muestra los indicadores importantes del estado y las características del tráfico VXLAN. Uno de los bits más relevantes es aquel que confirma que el identificador de red virtual (VNI) está activo, lo cual habilita la segmentación del tráfico dentro de la infraestructura compartida. Por otro lado, el indicador **Don't Learn**, configurado como False, indica que el aprendizaje automático de direcciones MAC está habilitado, permitiendo que los dispositivos dentro de la red actualicen dinámicamente sus tablas de encaminamiento en función del tráfico observado.

Además, el campo denominado "**Policy Applied**" tiene un valor de False, lo que indica que no se han aplicado políticas específicas al tráfico encapsulado en los paquetes VXLAN. Esto significa que no existen configuraciones adicionales relacionadas con el control de acceso, segmentación o reglas específicas para ese flujo de datos. Este enfoque demuestra que el protocolo puede funcionar de manera eficiente incluso sin configuraciones avanzadas de políticas.

El **VXLAN Network Identifier (VNI)** es un valor de 24 bits, que en el ejemplo correspondiente tiene el valor de 200. Este identificador permite la coexistencia de múltiples redes virtuales independientes dentro de la misma infraestructura física (red subyacente).

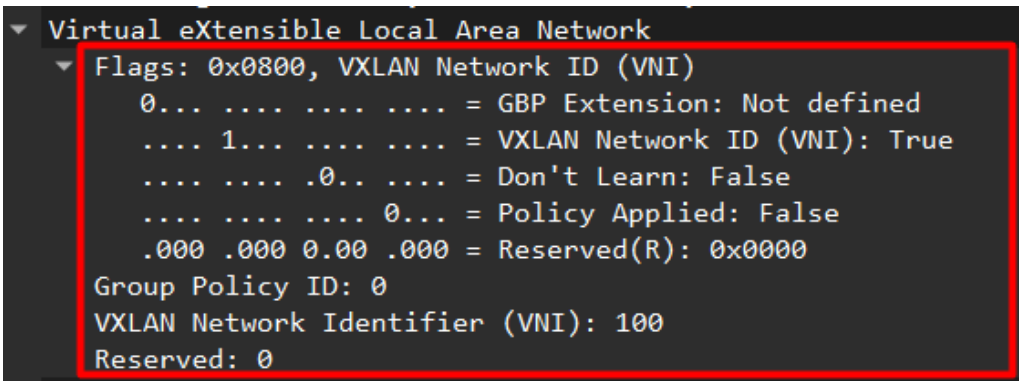
Específicamente, el valor de 200 en el VNI asigna que el paquete pertenece a una red virtual única, asegurando el aislamiento del tráfico de otras redes virtualizadas en el mismo escenario. La capacidad de segmentar redes a través de VNIs es una de las características clave de VXLAN, puesto que proporciona escalabilidad y flexibilidad a los entornos de redes definidas por software (SDN) y centros de datos modernos.

Por último, el paquete muestra un campo **Group Policy ID** con un valor de 0, lo que confirma que no se han asociado políticas de grupo con este paquete dentro de la red. Esto refuerza la simplicidad del flujo de tráfico en este caso específico, ya que no se aplican reglas adicionales para el control o priorización del tráfico. De igual manera, el campo **Reserved,** también configurado como 0, no contiene información adicional, lo cual es típico en implementaciones estándar del protocolo.

Elementos adicionales como **Don't Learn** o **Policy Applied** no están definidos en el EFC, pero forman parte de implementaciones específicas o extensiones del protocolo, comunes en entornos modernos. Sin embargo, estas no contradicen el RFC, ya que el estándar permite flexibilidad en la implementación, siempre y cuando se respeten los campos básicos.

Figura 33

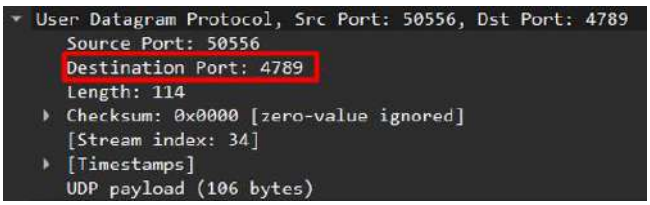
Header VXLAN – Tráfico VXLAN en Red eXtensible.



La **Figura 34** muestra el uso del protocolo UDP como el medio de transporte para VXLAN sobre la red Física, utilizando el puerto estándar 4789. Además, hay que aclarar que el puerto origen no posee relevancia, pues es un puerto aleatorio.

Figura 34

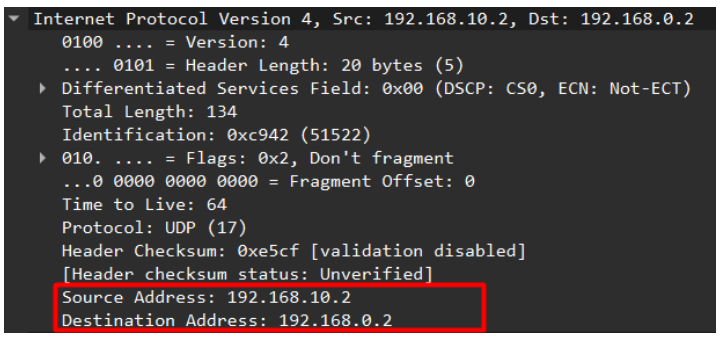
UDP Header.



La **Figura 35** indica el encabezado IP de los túneles VTEPs por los que atraviesa el encapsulamiento del tráfico para el transporte de los paquetes entre los hosts virtuales ubicados en distintas zonas de la red física, demostrando la capa 3 Underlay.

Figura 35

Outer IP header (capa 3 Underlay).



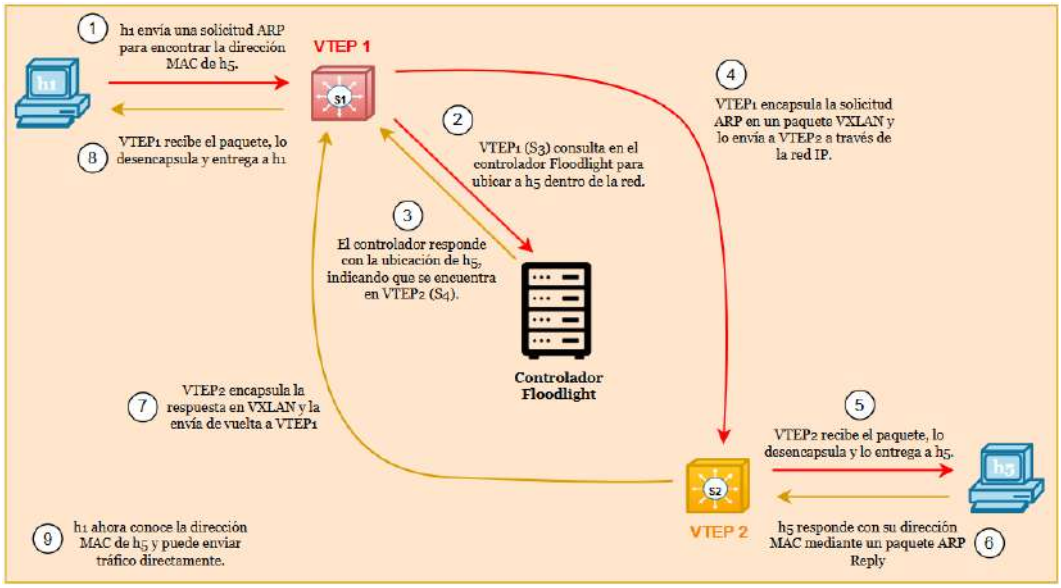
4.2.3 Diagrama de funcionamiento

En una infraestructura VXLAN con SDN, los switches VTEP (VXLAN Tunnel Endpoints) permiten que máquinas virtuales en diferentes redes físicas se comuniquen como si estuvieran en la misma LAN, utilizando túneles VXLAN sobre una red de capa 3. VXLAN encapsula el tráfico Ethernet dentro de un paquete UDP, mientras que el controlador SDN centraliza la gestión, optimizando la distribución del tráfico y eliminando la necesidad de protocolos de aprendizaje tradicionales. En lugar de depender de broadcast para descubrir direcciones MAC, el controlador comprende una base de datos global y realiza la configuración dinámicamente de las reglas de reenvío en los switches. Este enfoque híbrido permite escalar

redes virtualizadas, reducir la sobrecarga de tráfico y mejorar la automatización, facilitando la movilidad de cargas de trabajo en entornos como centros de datos y nubes**.**

Figura 36

Flujo de comunicación entre hosts del sistema.



Nota: El diagrama ilustra cómo se utiliza VXLAN para extender la conectividad de capa 2 sobre una infraestructura de red IP, y cómo el controlador SDN facilita la localización de los hosts dentro de la red.

Para identificar la funcionalidad de los túneles VXLAN, se utiliza un analizador de tráfico (Wireshark) en el escenario 2 (**Figura 6**), obteniendo resultados que confirman su comportamiento. Las pruebas realizadas muestran que, aunque VXLAN mantiene la comunicación basada en direcciones MAC, encapsula estos paquetes dentro de una red IP, lo que permite la interconexión entre dispositivos a través de múltiples dominios de capa 3. Este mecanismo garantiza que los dispositivos en diferentes subredes puedan comunicarse como si estuvieran en la misma red de capa 2, validando así la efectividad de VxLAN en la extensión de redes virtuales sobre infraestructuras IP, como se demuestra en las **Figuras 37, 38, 39, 40, 41 y 42.**

- Encapsulamiento entre VTEP 1 🔴 y VTEP 2 🟡

Figura 37

Encapsulamiento de VTEP 1 a VTEP 2.

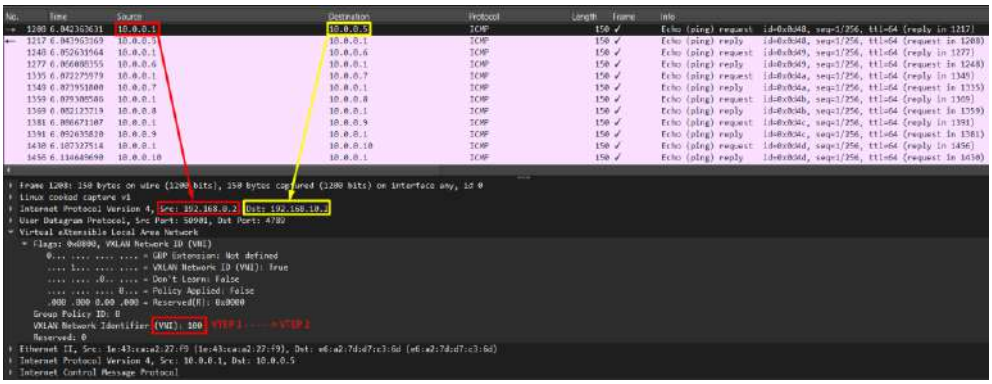
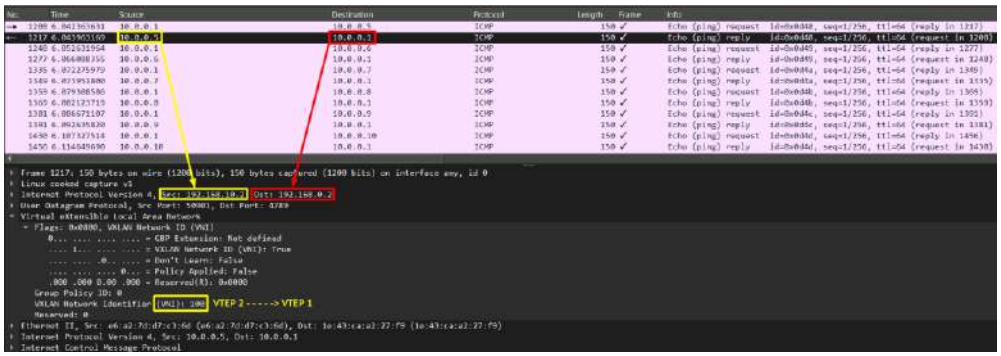


Figura 38

Encapsulamiento de VTEP 2 a VTEP 1.



- Encapsulamiento entre VTEP 1 🔵 y VTEP 2 🟡

Figura 39

Encapsulamiento de VTEP 3 a VTEP 2.

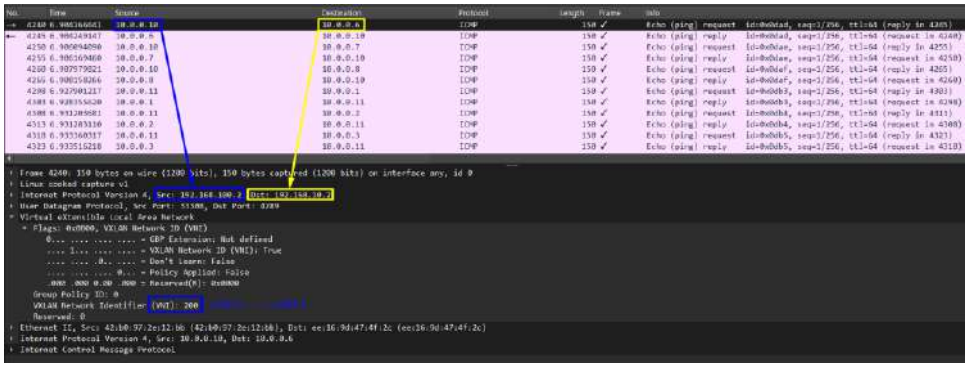
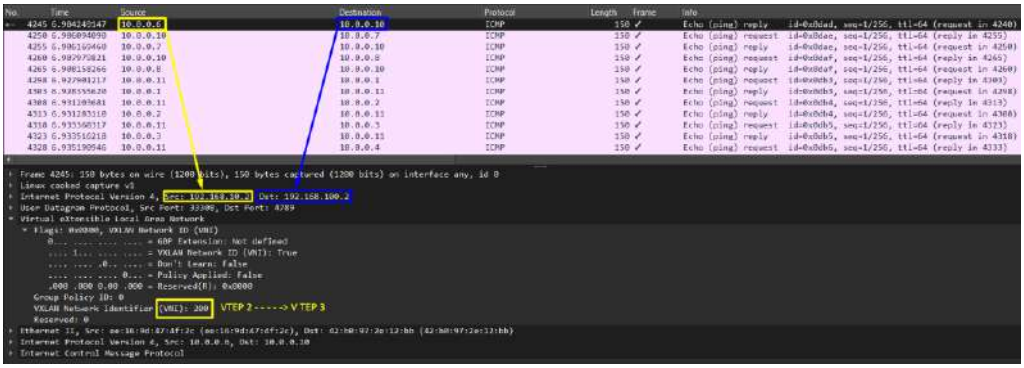


Figura 40

Encapsulamiento de VTEP 2 a VTEP 3.



- Encapsulamiento entre VTEP 1 🔴 y VTEP 3 🔵

Figura 41

Encapsulamiento de VTEP 1 a VTEP 3.

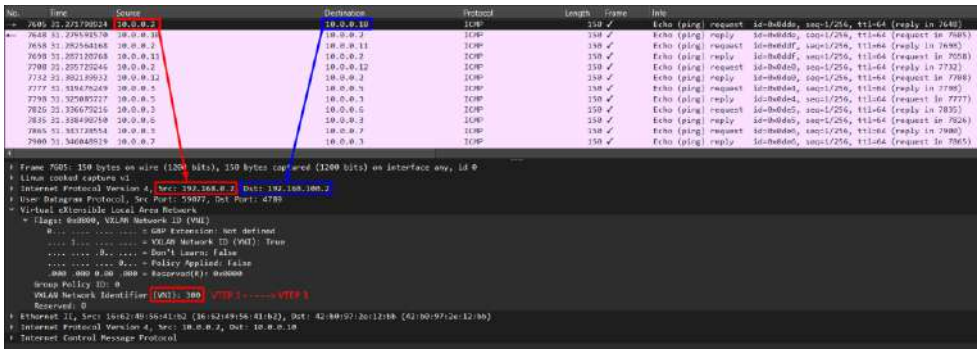
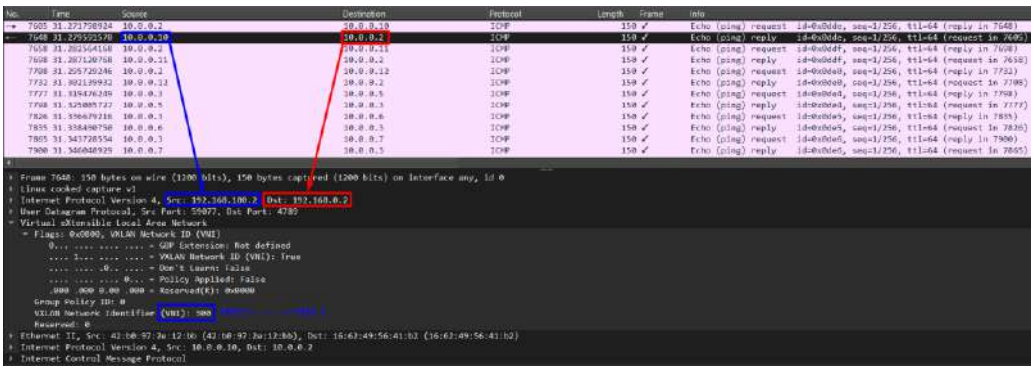


Figura 42

Encapsulamiento de VTEP 3 a VTEP 1.



4.2.4 Paquetes Perdidos

Los paquetes que aparecen como "no response found" en Wireshark como en la **Figura 43**, indican que se enviaron solicitudes de ping (ICMP Echo Request) a los destinos correspondientes, pero estos no respondieron con un ICMP Echo Reply. Las razones por la que podría darse este caso son:

- **Tiempo de Vida (TTL) y Reenvío en Túneles VXLAN**:

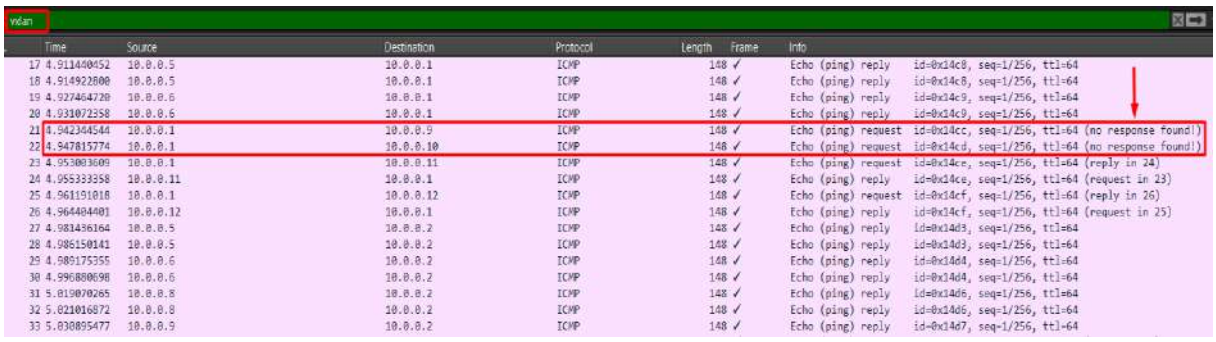
Al encapsular ICMP en UDP/VXLAN, el paquete ICMP original depende del encabezado UDP/IP para su reenvío. Si el TTL del paquete de capa 3 se agota en algún punto del túnel VXLAN, el paquete puede descartarse sin llegar al destino final, lo que podría explicar la falta de respuesta.

- **UDP es un Protocolo No Confiable:**

A diferencia de TCP, UDP no garantiza la entrega de los paquetes, ya que no cuenta con mecanismos de confirmación ni de retransmisión en caso de pérdida. Esto significa que, si un paquete UDP (en este caso, un paquete VXLAN) se pierde o es descartado en la red, no se realizará una retransmisión automática. Como resultado, las solicitudes ICMP encapsuladas en VXLAN podrían no llegar a su destino, lo que provocaría una falta de respuesta.

Figura 43

Paquete ICMP sin Respuesta.



4.3 Evaluación del Rendimiento y Conectividad en la Infraestructura SDN-VXLAN

Con el propósito de evaluar el desempeño de la infraestructura de red SDN-VXLAN implementada, presentada en la **Figura 6**, y a fin de validar la funcionalidad y el rendimiento de la infraestructura definida por software, se diseña una batería de pruebas[4] que permite observar el comportamiento de diversos indicadores clave de desempeño (KPIs) de la red.

Los parámetros considerados incluyen el rendimiento TCP (throughput), el tiempo de ida y vuelta (RTT), el balanceo de carga, la respuesta ante condiciones de congestión y la capacidad de adaptación de la red a eventos dinámicos de tráfico.

Para llevar a cabo esta evaluación, se toma como referencia principal la recomendación ITU-T Y.1540, que establece directrices para evaluar el rendimiento y la calidad en redes IP. Además, se utiliza la RFC 6349 como complemento, específicamente para evaluar el rendimiento de los flujos TCP.

- **Planteamiento de pruebas conforme a ITU-T Y.1540 y RFC 6349**

La recomendación **ITU-T Y.1540** proporciona directrices para diseñar procedimientos de medición que permiten analizar parámetros críticos de desempeño en redes IP, tanto en condiciones normales como degradadas. Siguiendo esta normativa, se presenta un conjunto de pruebas específicas para evaluar la red SDN-VXLAN desplegada, con un enfoque en aspectos como el retardo, la eficiencia de transmisión y la distribución del tráfico en condiciones variables. Este proceso se detalla a continuación:

- **Paso 1: Definir el esquema de red para las pruebas.**

Se utiliza una topología spine-leaf con túneles VXLAN, gestionada mediante un controlador SDN Floodlight, la cual conecta múltiples hosts distribuidos en distintos dominios

[4] Batería de pruebas: conjunto estructurado de pruebas diseñadas para evaluar el funcionamiento, el rendimiento o la calidad de un sistema en diferentes condiciones y escenarios

VNI. Los puntos de medición se definieron entre los hosts finales (h2–h12) y el servidor central h1, actuando este último como nodo receptor en la mayoría de los experimentos. También se considera la captura de estadísticas en switches intermedios (s1 a s5) para validar el comportamiento del plano de datos.

- **Paso 2: Establecimiento de un conjunto de KPIs**

Se seleccionan los siguientes indicadores clave para medir el rendimiento de la red, todos alineados con los parámetros definidos por ITU-T Y.1540 y complementados por RFC 6349, tal como se muestra en la **Tabla 6**.

Tabla 6

Parámetros definidos para pruebas implementadas.

Prueba	Normativa	Fundamento Teórico
Tráfico TCP con eventos	Según ITU-T Y.1540 (Sección 5.4 y 5.5)	Evaluación del comportamiento de la red bajo tráfico fluctuante y dinámico, y observación de la recuperación de eventos.
Medición de RTT	Según ITU-T Y.1540 (Sección 6.2) e ITU-T G.1010	Medición de RTT con ping y múltiples muestras. Los valores deben estar muy por debajo del umbral de 150 ms para servicios interactivos según la normativa.
Throughput TCP	Según RFC 6349	Utilización de iperf, tal como propone la RFC. Evaluación de flujos TCP reales, con sesiones de duración controlada para posterior observación de eficiencia según la ubicación (intra/inter VTEP).

Balanceo de carga	Según ITU-T Y.1540 (Sección 6.11)	Evaluación de si la distribución de tráfico en los switches es equitativa. La norma no habla de “balanceo” explícitamente, pero sí de distribución eficiente para evitar degradación.
Congestión y Laberio	Según ITU-T Y.1540 (Sección 6.12)	Simulación de una condición degradada (congestión) y medición de la mejora al aplicar un algoritmo de corrección.

Nota: Las herramientas utilizadas incluyen iperf para pruebas de rendimiento, ping para medición de latencia, tcpdump para capturas de tráfico, Wireshark para análisis gráfico, y ovs-ofctl para extracción de estadísticas en switches Open vSwitch.

- **Paso 3: Definir escenarios específicos para medir los KPIs establecidos.**

Se definen escenarios específicos para medir los KPIs establecidos. Se diseñan cinco pruebas distintas que permiten observar el comportamiento de la red ante situaciones clave:

1. Simulación de tráfico TCP con eventos temporizados.
2. Medición de RTT desde cada host hacia h1.
3. Evaluación de throughput TCP.
4. Medición del impacto del balanceo de carga.
5. Simulación de congestión y análisis de la efectividad del algoritmo Laberio.

- **Paso 4: Presentar un resumen de los resultados obtenidos.**

En los apartados siguientes se analizan detalladamente los resultados obtenidos para cada prueba, asociándolos con los KPIs medidos y validándolos conforme a los umbrales,

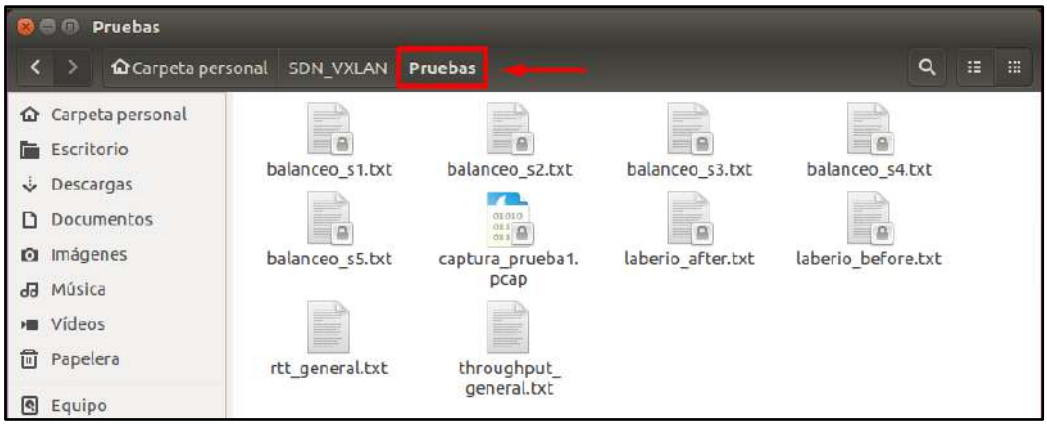
metodologías y lineamientos definidos por las normativas internacionales ITU-T Y.1540 y RFC 6349. La interpretación incluye análisis gráfico, cálculos matemáticos, así como observación del comportamiento de la red bajo eventos de carga, congestión o redistribución según sea el caso de la prueba en cuestión.

Desde un enfoque técnico, todas las pruebas están diseñadas para evaluar parámetros como la latencia, la pérdida de paquetes y la eficiencia en el enrutamiento entre VNIs. Aunque los resultados de cada ejecución muestran pequeñas variaciones, estas fluctuaciones son mínimas y propias de un entorno emulado. Por ello, se ha considerado una ejecución representativa, lo que aporta información valiosa para una evaluación integral del escenario de red, centrándose en un patrón general de respuesta. Estos datos permiten destacar la robustez, escalabilidad y adaptabilidad de la infraestructura, proporcionando una base sólida para validar la viabilidad de implementar soluciones similares en entornos de producción reales.

Además, las pruebas se manejan y ejecutan de forma programada, aprovechando las capacidades de automatización que brinda la arquitectura SDN, destinando un espacio específico dentro de la VM para el almacenamiento de los archivos generados, los cuales se utilizan posteriormente para el análisis de los resultados. La **Figura 44** muestra la ubicación de la carpeta que contiene los resultados, los cuales son cruciales para la evaluación del rendimiento de la red, validar su funcionalidad y beneficios.

Figura 44

Carpeta de resultados generados por las pruebas realizadas.



*4.3.1 **PRUEBA 1: Simulación de tráfico TCP con eventos temporizados.***

La prueba tiene como objetivo simular una carga progresiva de tráfico TCP en la red, siguiendo un esquema de eventos distribuidos en el tiempo, como se detalla en la **Tabla 7**. El propósito es evaluar la capacidad de la infraestructura para gestionar un tráfico en aumento, identificando posibles degradaciones en la conectividad o el rendimiento a medida que más hosts inician la transmisión de datos. A continuación, se representa en la **Figura 45**, la metodología que se realiza para la ejecución de la prueba propuesta.

Figura 45

Metodología para la simulación de tráfico TCP progresivo.

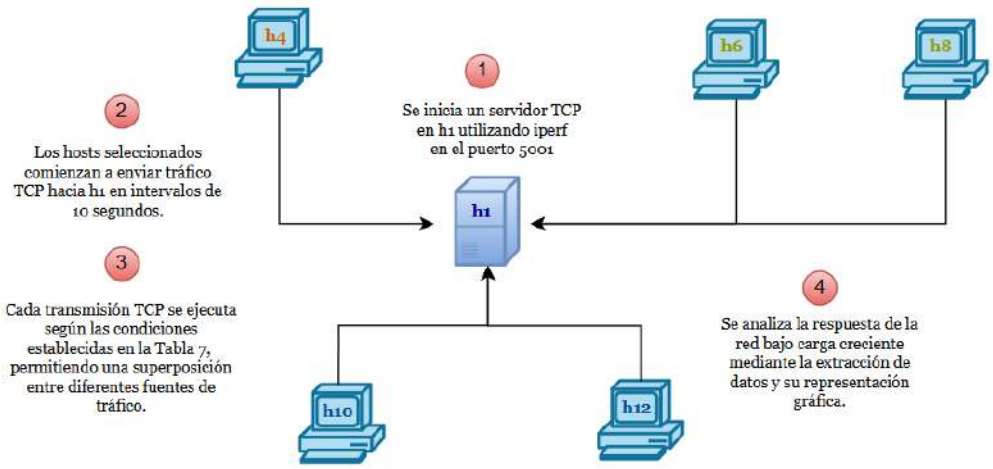


Tabla 7

Condiciones para el proceso de simulación de tráfico TCP.

Tiempo (s)	Acción
0	h4 inicia tráfico hacia h1
10	h6 inicia tráfico hacia h1
20	h8 inicia tráfico hacia h1
30	h4 detiene tráfico
30	h10 inicia el tráfico hacia h1
40	h4 reanuda tráfico
40	h12 inicia tráfico hacia h1
50	Fin del tráfico

La **Tabla 8** presenta los parámetros del comando de ejecución utilizados para llevar a cabo la prueba de tráfico TCP progresiva con la herramienta iperf. En este caso, se emplea el comando `<host_cliente> iperf -c 10.0.0.1 -p 5001 -t 50 &`. Este tipo de prueba es comúnmente empleada para evaluar el rendimiento de la red, midiendo el ancho de banda disponible, la latencia y la calidad de la conexión entre un cliente y un servidor.

Tabla 8

Descripción de los parámetros para prueba de tráfico TCP progresiva.

Comando	Función
(h4, h6, h8, h10, h12)	Host cliente
iperf	Ejecuta la herramienta iperf (versión 2)
-c <IP_SERVIDOR>	Indica que es cliente y se conecta a la IP del servidor
-p 5001	Usa puerto por defecto de iperf versión 2

-t 50	Ejecuta la prueba durante 50 segundos
&	Ejecuta el comando en segundo plano.

Para analizar los resultados de cada prueba, se implementa un sistema automático para guardar los datos obtenidos, ajustado según el tipo de observación requerida. En este caso, se realiza una captura de paquetes TCP con Wireshark durante 50 segundos, respaldada por la herramienta IO Graphs, como se muestra en la **Figura 46.** Durante este intervalo, se observa la interacción entre los equipos participantes, según la **Tabla 7**. La gráfica generada permite visualizar y analizar el comportamiento del tráfico y la transferencia de datos a lo largo del tiempo, lo cual es clave para diagnosticar problemas, medir el rendimiento y validar la eficiencia de la red o protocolo en prueba.

Además, todos los datos crudos generados durante el evento se almacenan en un archivo de texto. De estos, se extraen las métricas más relevantes, como el total de bytes, el total de paquetes y el tamaño promedio, para un análisis más detallado (archivo de métricas por segundo). Estos datos sintetizados, extraídos del archivo y presentados en la **Tabla 9**, son esenciales para el análisis cuantitativo y garantizan una validación rigurosa de la prueba según la normativa vigente.

Tabla 9

Datos extraídos de Wireshark de la simulación de tráfico TCP por intervalo de tiempo.

Tiempo (s)	Total Bytes	Total Paquetes	Tamaño Promedio Paquete (bytes)
0	344316836	7713	44641,1
1	518149048	10186	50868,75
2	513953312	9228	55694,98
3	523948028	10089	51932,6
4	522913144	10484	49877,26
5	431530024	7640	56482,99
6	505437244	9465	53400,66

7	503437488	9330	53959
8	513390748	9967	51509,05
9	585607584	11250	52054,01
10	408418328	8218	49698,02
11	437955960	8608	50877,78
12	423558924	8001	52938,25
13	422968504	8154	51872,52
14	426351760	7802	54646,47
15	381024640	7300	52195,16
16	514163048	9950	51674,68
17	403420716	7757	52007,31
18	432881000	8220	52661,92
19	388969508	7563	51430,58
20	441644816	8754	50450,63
21	400348224	7872	50857,24
22	423406420	8535	49608,25
23	603157888	12264	49181,17
24	537243460	11271	47666
25	486640340	9793	49692,67
26	342091000	7136	47938,76
27	377446540	7799	48396,79
28	407222432	8066	50486,29
29	465395584	9356	49743,01
30	421532344	10120	41653,39
31	549681992	11908	46160,73
32	544305132	11229	48473,16
33	596118168	12372	48182,85
34	490841252	10267	47807,66
35	568178404	11991	47383,74
36	549345580	11571	47476,07
37	565998964	11695	48396,66
38	516424392	10904	47361
39	615743140	12643	48702,3
40	419034424	9426	44455,17
41	374053376	8260	45284,91

42	452733528	9232	49039,59
43	411373084	8785	46826,76
44	427410760	8768	48746,67
45	462332500	9103	50789,03
46	474612764	9327	50885,9
47	511090940	10061	50799,22
48	336075576	7100	47334,59
49	417089032	8470	49243,1
50	224474452	4469	50229,24

- **Análisis en base a la ejecución de la prueba**

Durante la simulación, el tráfico TCP se genera de forma progresiva desde distintos nodos hacia un mismo destino, comenzando con el nodo h4 en el segundo 0, seguido por los nodos h6 y h8 en los segundos 10 y 20 respectivamente. Este incremento gradual de la carga refleja un escenario realista donde múltiples fuentes transmiten simultáneamente, permitiendo evaluar el desempeño de la red bajo una demanda creciente.

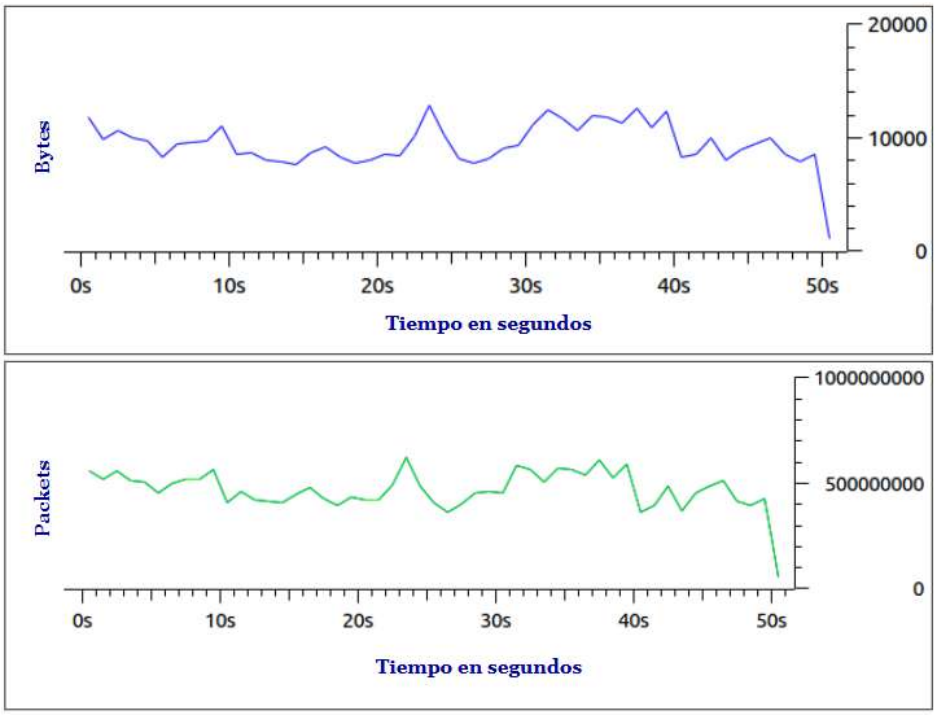
A partir del segundo 30, la prueba muestra eventos de detención y reanudación del tráfico, junto con la incorporación de nuevos nodos como h10 y h12, lo que genera fluctuaciones dinámicas en el flujo de datos. Estas variaciones se reflejan en las gráficas de IO Graphs de la Figura 46, donde el volumen total de bytes, el número de paquetes y el tamaño promedio de los mismos corresponden a los tiempos establecidos en la simulación.

A pesar de estas fluctuaciones, la red muestra una gestión eficiente del balanceo de carga. La caída temporal en el tráfico tras la detención de h4 se compensa rápidamente con el inicio del flujo desde h10, y la posterior reanudación de h4, junto con la activación de h12, contribuyen a estabilizar el tráfico. Esto demuestra que la red se adapta eficazmente a los cambios, manteniendo un flujo constante de datos. Las fluctuaciones en el tamaño promedio de

los paquetes, que oscilan entre 33,000 y 60,000 bytes, reflejan la dinámica del tráfico TCP bajo estas condiciones variables.

Figura 46

Representación de la evolución del tráfico TCP en función del tiempo.



- **Evaluación normativa**

De acuerdo con la Recomendación **ITU-T Y.1540**, el análisis del comportamiento de redes IP no debe limitarse a condiciones ideales. En las Secciones 5.4 y 5.5, se establece que deben evaluarse escenarios operativos bajo condiciones degradadas o variables, como las fluctuaciones en la carga de tráfico. Además, la Sección 6.11 señala que si una red se adapta eficientemente a estas variaciones sin sufrir una degradación sostenida, se considera que cumple con los requisitos de calidad establecidos por la norma.

Además, es importante resaltar que, el tiempo de 50 segundos para la prueba se establece para asegurar que la máquina virtual (VM) pueda manejar la captura de datos sin sobrecargar sus recursos. Este periodo es suficiente para analizar la respuesta de la red ante los eventos

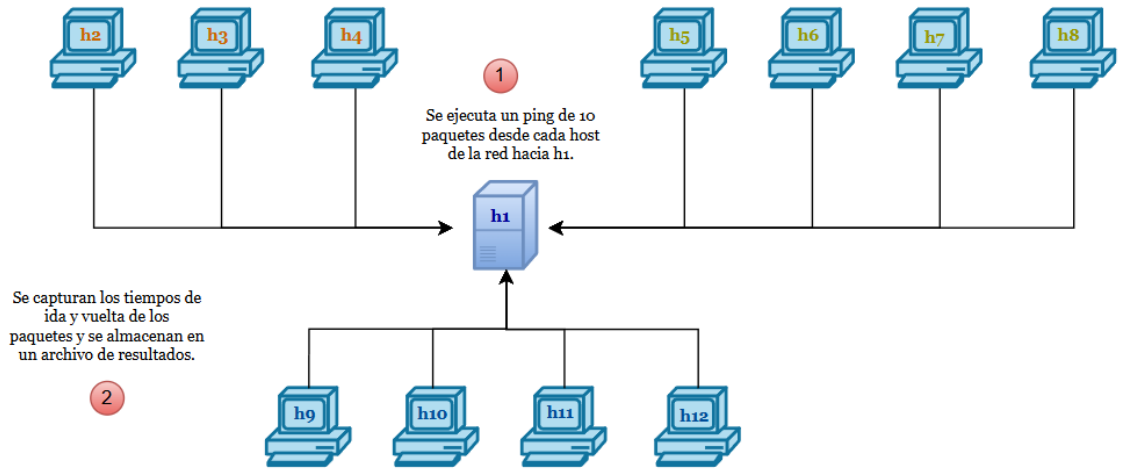
programados y las fluctuaciones del tráfico. Aunque la norma ITU-T Y.1540 recomienda un tiempo de 5 a 10 minutos (300 a 600 segundos) para obtener resultados más representativos.

4.3.2 PRUEBA 2: Medición de RTT (Tiempo de ida y vuelta)

La prueba tiene como objetivo la evaluación de la latencia de la red por medio de la medición del RTT (Round Trip Time), que representa el tiempo total que demora un paquete en viajar desde un host origen hasta su destino y regresar. La estabilidad y minimización de la latencia son aspectos clave en entornos SDN-VXLAN, dado que el tráfico debe atravesar múltiples túneles y dispositivos intermedios, lo que puede introducir retardos adicionales. Un RTT bajo y consistente es fundamental para garantizar un rendimiento óptimo en la comunicación dentro de la red. A continuación, en la **Figura 47**, se presenta la metodología utilizada en esta prueba que detalla el proceso empleado para el análisis de los valores de latencia.

Figura 47

Metodología para simulación de la prueba 2.



La **Tabla 10** muestra los parámetros del comando `<host_cliente> ping -c 10 h1`, que se utiliza para realizar la prueba de RTT (Round Trip Time). Esta prueba mide el tiempo que tarda un paquete en viajar desde el dispositivo origen hasta el destino y regresar. Esta

métrica es crucial para evaluar la latencia de la red y detectar posibles problemas de conectividad.

Tabla 10

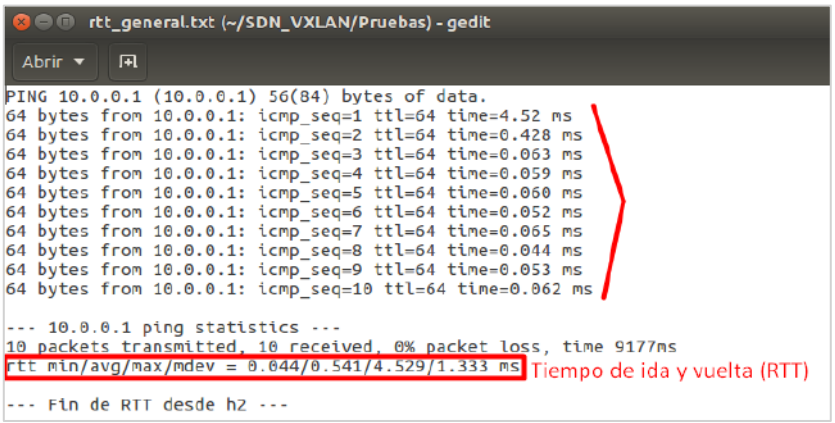
Parámetros del comando para prueba RTT.

Comando	Función
(h2, h3, h4, h5…h12)	Host cliente
ping	Lanza paquetes ICMP hacia un host para medir latencia
-c 10	Envía 10 paquetes
h1	Host destino

La **Figura 48** demuestra el proceso de recopilación de la información obtenida durante la prueba de tiempo de ida y vuelta (RTT). Los datos se almacenan automáticamente en un archivo de texto generado durante la ejecución de la prueba, donde todos los hosts participantes en la prueba siguen un esquema de visualización uniforme, y la información correspondiente a cada uno se unifica en un único documento general, lo que facilita el análisis integral de los resultados.

Figura 48

Evidencia de los resultados RTT en el archivo txt.



- **Análisis en base a la ejecución de la prueba 2**

La Prueba 2 evalúa el desempeño de la red en términos de latencia, a través de la medición del parámetro RTT (Round Trip Time). Este parámetro es primordial en redes que integran SDN-VXLAN, donde el tráfico atraviesa o pasa por múltiples switches, encapsulaciones (VXLAN), y dispositivos de capa 3 que pueden agregar retardo en la red.

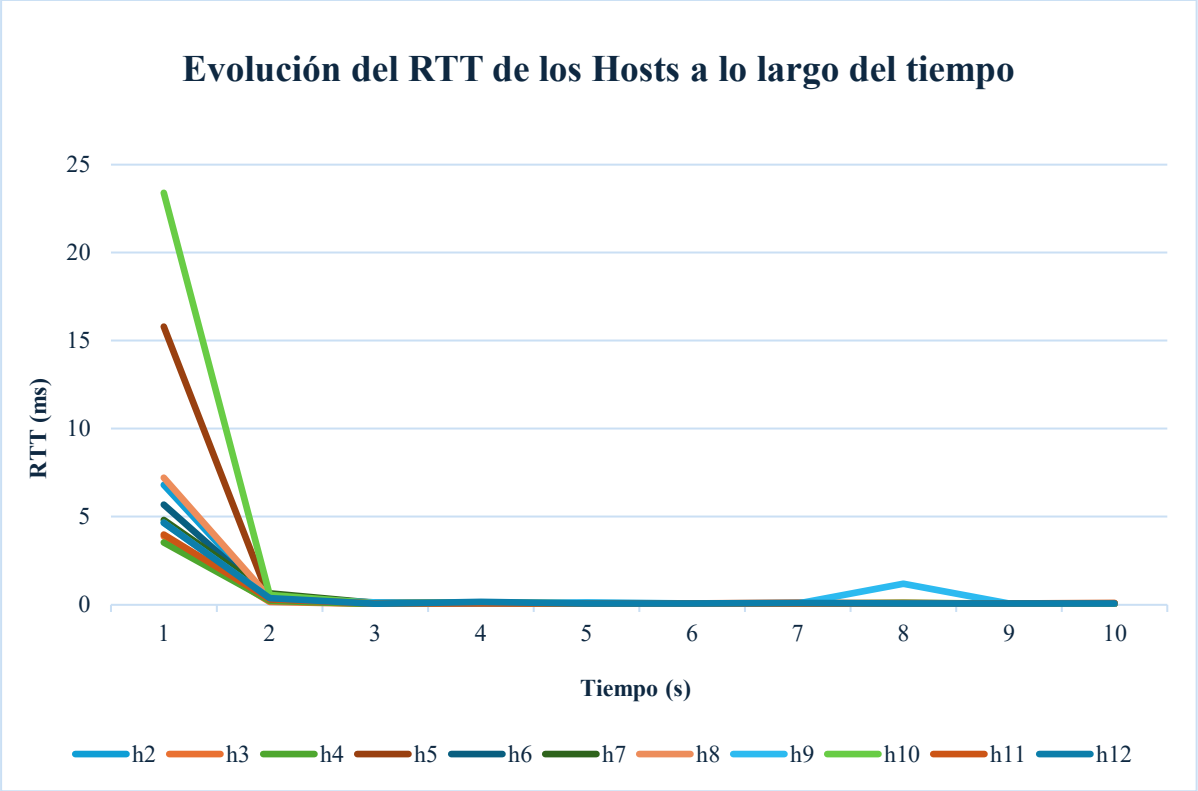
Para esto, se emplea el comando ping como herramienta de medición en concreto, enviando múltiples paquetes ICMP desde cada host (h2 - h12) hacia el nodo h1 (servidor destino). Los valores de RTT son analizados en función del tiempo y por host, lo que permitió obtener tanto la evolución temporal como el promedio por emisor.

El análisis muestra un comportamiento esperado en redes correctamente configuradas. Algunos hosts, como h9, h4 y h10, presentaron valores elevados de RTT en los primeros paquetes, lo cual se debe comúnmente a fenómenos como la resolución ARP, el inicio de sesiones o la activación de enlaces dormidos. Sin embargo, a partir del segundo 2, todos los valores disminuyeron considerablemente, estabilizándose por debajo de los 2 ms en la mayoría de los casos. Esto indica que la red maneja un RTT bajo, lo cual es beneficioso desde un punto de vista técnico y teórico.

También, se observan pequeñas fluctuaciones transitorias entre los segundos 7 y 9, asociadas a cambios en la ruta o congestión menor. Estas no comprometen la estabilidad del sistema, pero una opción debería corresponder a optimizar la red si es que fuera orientada a tráfico en tiempo real. A continuación, se presenta la **Figura 49**, que ilustra los comportamientos mencionados previamente, junto con los valores de la información extraída en una tabla para facilitar su visualización y comprender el análisis que se realiza.

Figura 49

Resultados de la simulación de la prueba 2 – Evaluación de RTT.



Nota: El análisis de estos resultados permite optimizar la configuración de los switches Open vSwitch (OVS) y del controlador SDN para minimizar la latencia en la red. Los datos representados en esta figura se encuentran en la **Tabla 11.**

Tabla 11

Valores de RTT por segundo para cada host.

Tiempo (s)	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11	h12
1	6,81	3,89	3,54	15,8	5,69	4,82	7,22	4,66	23,4	3,99	4,66
2	0,327	0,182	0,227	0,325	0,394	0,657	0,404	0,391	0,562	0,325	0,365
3	0,054	0,059	0,058	0,133	0,075	0,093	0,085	0,13	0,08	0,078	0,062
4	0,053	0,075	0,054	0,109	0,085	0,082	0,082	0,08	0,145	0,079	0,157

5	0,049	0,058	0,053	0,078	0,088	0,087	0,104	0,131	0,087	0,086	0,078
6	0,083	0,056	0,079	0,072	0,083	0,083	0,087	0,074	0,069	0,086	0,091
7	0,066	0,055	0,084	0,07	0,09	0,08	0,144	0,073	0,082	0,072	0,098
8	0,049	0,103	0,05	0,106	0,083	0,053	0,091	1,2	0,121	0,075	0,088
9	0,052	0,056	0,062	0,064	0,085	0,077	0,076	0,074	0,073	0,08	0,076
10	0,052	0,057	0,065	0,076	0,088	0,077	0,084	0,075	0,076	0,101	0,075

Nota: Estos valores representados en la tabla corresponden al Round Trip Time (RTT) medido en unidades de milisegundos (ms) para cada uno de los 10 intervalos de tiempo establecidos a lo largo de la prueba.

- **Evaluación normativa**

La Recomendación **ITU-T Y.1540**, en su Sección 6.2, específica que el RTT es uno de los parámetros esenciales para evaluar el desempeño de la transferencia IP. La medición debe realizarse por medio de pruebas activas (como ICMP/ping), en condiciones representativas del entorno de red.

Por su parte, la recomendación complementaria de la normativa ITU-T G.1010 manifiesta que para servicios interactivos (como voz o aplicaciones en tiempo real), el valor referencial máximo para el RTT es de 150 ms. En este estudio, los RTT promedio se ubicaron muy por debajo del umbral indicado, con el 100% de los valores inferiores a los 10 ms.

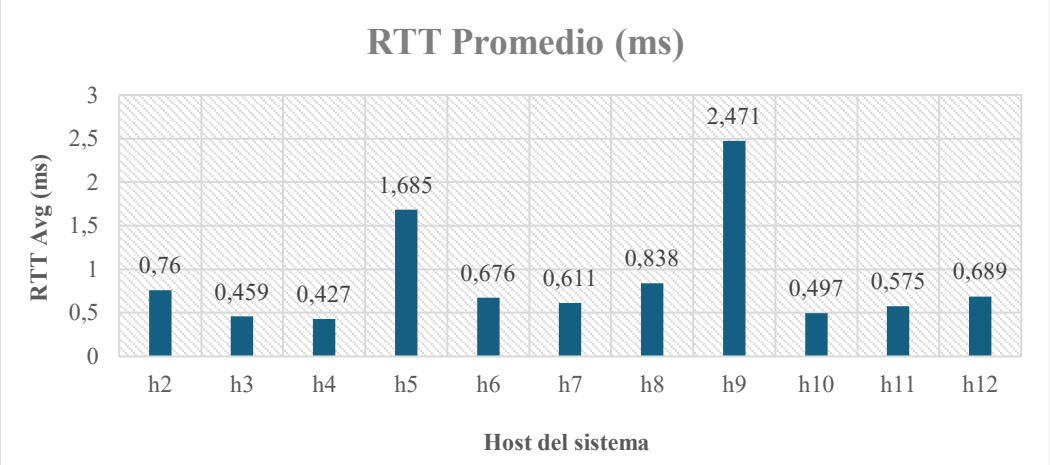
$$\boldsymbol{RTT_{prom}} = \left(\frac{\mathbf{1}}{\boldsymbol{N}}\right) \times \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{RTT_i} \qquad \textit{(Ec.1)}$$

Por ende, se concluye que la infraestructura de red cumple con los lineamientos normativos tanto de ITU-T Y.1540 como de G.1010, en términos de latencia promedio, estabilidad temporal, y ausencia de retardo excesivo. Evidenciando negación en síntomas de cogestión crítica o pérdida de paquetes.

Es así como, en base a la **Ec.1**, se obtienen los valores promedios de la **Figura 50**, que demuestran los promedios de RTT, donde sobresalen h4 con 0.482 ms como el más bajo y h11 con 6.638 ms como el más elevado. Aun así, todos los valores permanecen dentro de márgenes aceptables para entornos IP estándar según la normativa como se indicó anteriormente.

Figura 50

RTT promedio obtenido en la prueba 2.

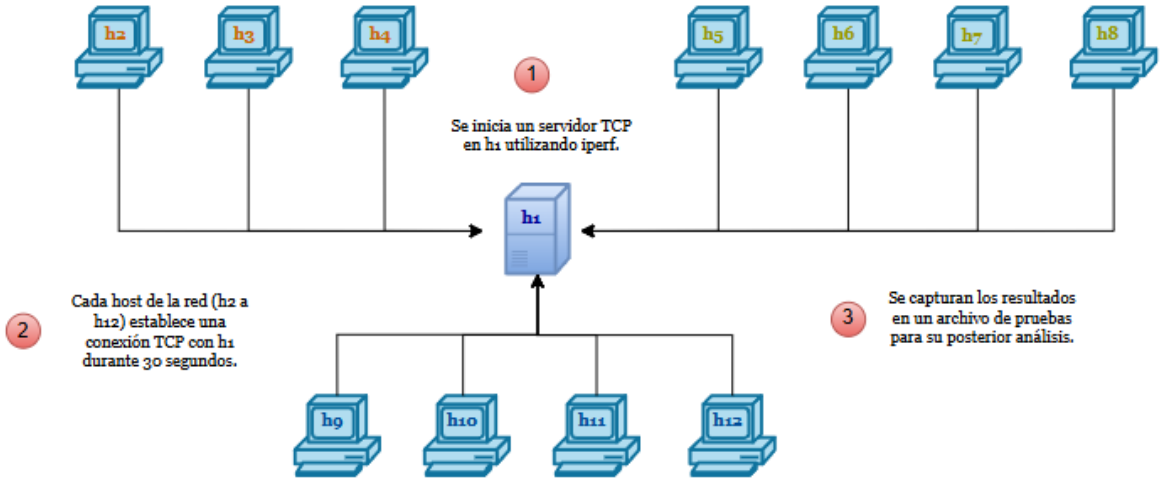


*4.3.3 **PRUEBA 3: Medición de rendimiento (Throughput).***

El objetivo de la prueba es evaluar la capacidad de transmisión de datos de la red en términos de throughput (Mbps), un parámetro clave para determinar la eficiencia y el desempeño de la red. Un valor elevado de throughput indica que la red puede manejar grandes volúmenes de tráfico sin experimentar una degradación en el rendimiento, lo cual es esencial en entornos de alta demanda. A continuación, la **Figura 51** detalla la metodología establecida para la ejecución de la prueba, describiendo los procedimientos utilizados para medir y analizar el throughput en la red.

Figura 51

Metodología para simulación de la prueba 3.



La **Tabla 12** representa cada parámetro del comando `<host_cliente> iperf -c 10.0.0.1 -p 5001 -t 30`, que se emplea para realizar la prueba de throughput (rendimiento de red) utilizando iperf como herramienta. Este comando configura al cliente iperf para enviar datos de manera continua durante 30 segundos al servidor h1 (IP 10.0.0.1) a través del puerto 5001, permitiendo medir el volumen total de datos transferidos y calcular la tasa promedio de transferencia, conocida como throughput. Estas pruebas son indispensables cuando se quiere evaluar la capacidad real de transmisión entre el cliente y el servidor, proporcionando una estimación exacta del ancho de banda efectivo disponible en la conexión de red.

Tabla 12

Descripción de los parámetros para la prueba de Throughput.

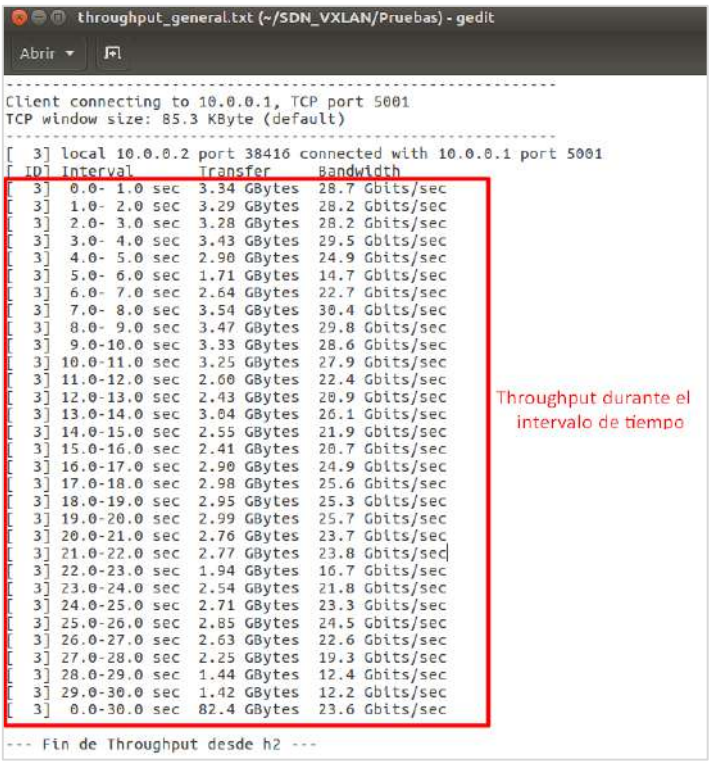
Comando	Función
(h2, h3, h4, h5…h12)	Host cliente
iperf	Ejecuta la herramienta iperf (versión 2)
-c <IP_SERVIDOR>	Indica que es cliente y se conecta a la IP del servidor

-p 5001	Usa puerto por defecto de iperf versión 2
-t 30	Ejecuta la prueba durante 30 segundos

De igual manera, para esta prueba se configura el guardado automático de los valores resultantes, tal como se muestra en la **Figura 52**. En esta figura, se observa claramente la transferencia de datos TCP entre el cliente y el servidor en cada intervalo. Este formato se aplica a cada host involucrado y los datos se guardan progresivamente en el mismo archivo txt.

Figura 52

Evidencia de los resultados del throughput en el archivo txt.



- **Análisis en base a la ejecución de la prueba**

Los resultados obtenidos para el throughput, ilustrados en la **Figura 53** y presentados en la **Tabla 13,** muestran una diferencia notable entre los hosts h2, h3 y h4 (mismo VTEP que h1), que alcanzaron hasta 30 Gbps, y los hosts h5-h12 (otros VTEPs), cuyo throughput se

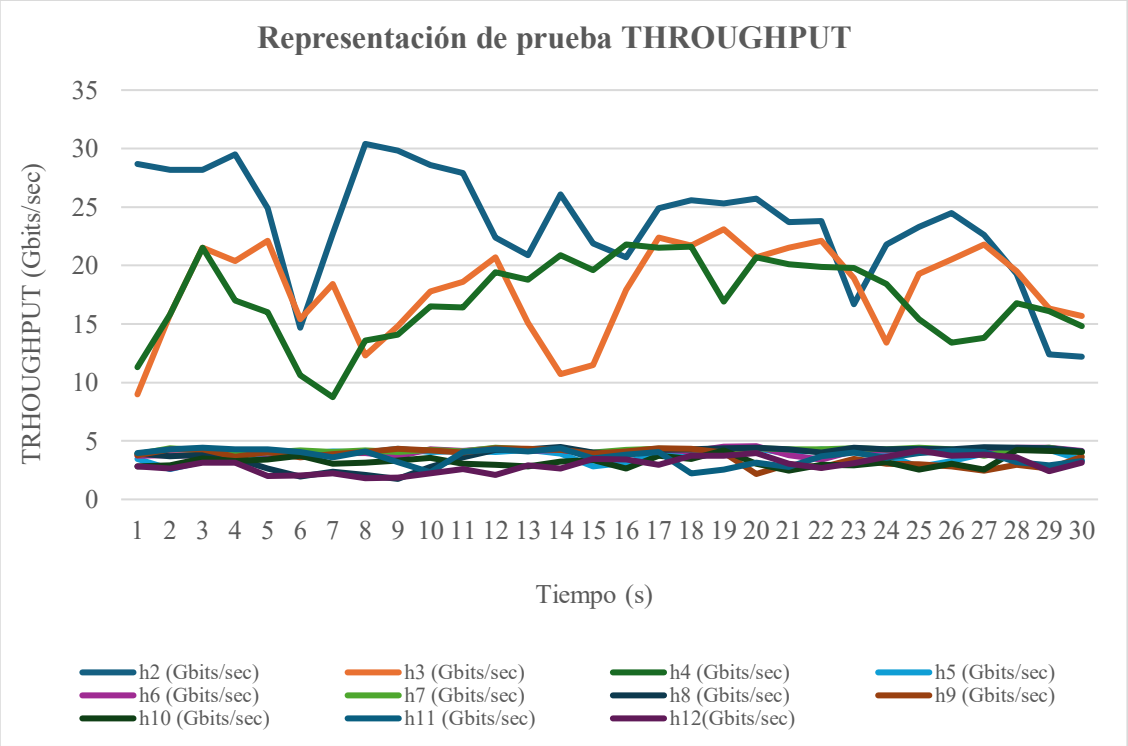
mantuvo por debajo de 5 Gbps. Esta diferencia se explica por la necesidad de encapsulación VXLAN y el tránsito por la infraestructura spine-leaf, lo cual introduce sobrecarga de red y mayor latencia.

Aunque los hosts h5-h12 muestran un throughput inferior en comparación con los hosts h2, h3 y h4, es importante destacar que todos los nodos están funcionando dentro de los parámetros esperados para una red de este tipo. A pesar de las fluctuaciones y menores valores de throughput, la red demuestra una capacidad adecuada para manejar tráfico en diferentes VTEPs, lo que refleja una estructura flexible y escalable. Además, el hecho de que los hosts h5-h12 mantengan un rendimiento constante, incluso si es más bajo, sugiere que la red es capaz de operar de manera confiable bajo condiciones de carga variada, lo cual es un aspecto positivo para su estabilidad a largo plazo.

El throughput que se observa está dentro del rango adecuado, debido a que, cumple con las directrices de establecidas en el RFC 6349, pues, en mayoría los hosts presentan un rendimiento consistente y eficiente, lo que es un buen indicador de la calidad de la red. Sin embargo, se podría investigar más a fondo las fluctuaciones en algunos hosts para asegurar que la red mantiene un rendimiento óptimo en todo momento.

Figura 53

Resultados de la simulación de la prueba 3.



Nota: Los resultados de esta prueba presentados en valores en la **Tabla 13**, permiten evaluar si la red está funcionando dentro de los valores esperados o si es necesario optimizar la gestión de paquetes en OVS y OpenFlow.

Tabla 13

Valores de Throughput por segundo para cada host expresados en Gbps.

Tiempo (s)	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11	h12
1	28,7	8,98	11,3	3,44	3,75	3,85	3,91	3,79	2,81	3,97	2,84
2	28,2	15,8	15,8	2,59	3,8	4,36	3,68	4,27	2,89	4,29	2,64
3	28,2	21,5	21,5	3,25	3,73	4,21	3,83	4,28	3,5	4,42	3,12
4	29,5	20,4	17	4,04	4,12	4,09	3,5	3,62	3,28	4,26	3,14
5	24,9	22,1	16	3,61	3,87	4,02	2,65	4,03	3,4	4,28	2,02
6	14,7	15,4	10,6	3,89	4,06	4,18	1,95	3,58	3,74	4,05	2,04
7	22,7	18,4	8,74	4,11	3,97	4,07	2,37	3,85	3,04	3,61	2,24
8	30,4	12,3	13,6	4	3,98	4,17	2,08	4,06	3,16	4,09	1,81
9	29,8	14,8	14,1	4,32	3,67	4,03	1,75	4,31	3,32	3,2	1,88
10	28,6	17,8	16,5	4,07	4,29	4,24	2,77	4,18	3,57	2,31	2,21
11	27,9	18,6	16,4	4,14	4,15	4,1	3,61	4	3,03	4,04	2,57

12	2,4	20,7	19,4	4,07	4,36	4,42	4,22	4,36	2,97	4,26	2,11
13	20,9	15,1	18,8	4,23	4,16	4,27	4,27	4,33	2,82	4,12	2,93
14	26,1	10,7	20,9	3,9	4,22	4,26	4,48	4,23	3,25	4,37	2,66
15	21,9	11,5	19,6	2,83	3,85	4,01	4	3,92	3,36	3,51	3,46
16	20,7	17,9	21,8	3,12	4,06	4,24	3,91	4,11	2,66	3,82	3,41
17	24,9	22,4	21,5	3,98	4,26	4,33	4,22	4,37	3,73	4,05	2,96
18	25,6	21,7	21,6	3,6	4,12	4,27	4,28	4,32	3,47	2,22	3,75
19	25,3	23,1	16,9	4,01	4,49	4,15	4,39	4,01	4,21	2,54	3,75
20	25,7	20,7	20,7	4,35	4,55	4,36	4,42	2,17	3,03	3,13	3,97
21	23,7	21,5	20,1	3,92	3,8	4,27	4,29	3,02	2,46	2,79	3,03
22	23,8	22,1	19,9	4,3	3,4	4,3	4	2,75	2,97	3,68	2,68
23	16,7	18,9	19,8	4,02	4,16	4,39	4,41	3,46	2,93	4	3,08
24	21,8	13,4	18,4	3,77	4,21	4,29	4,29	3,07	3,17	3,55	3,64
25	23,3	19,3	15,4	2,78	4,26	4,42	4,34	3	2,53	3,96	4,18
26	24,5	20,5	13,4	3,27	4,11	4,29	4,3	2,8	3,06	4,14	3,75
27	22,6	21,8	13,8	3,91	4,09	3,73	4,46	2,45	2,56	4,03	3,84
28	19,3	19,5	16,8	4,34	4,4	4,22	4,42	2,96	4,22	3,19	3,58
29	12,4	16,3	16,1	4,22	4,4	4,4	4,36	2,65	4,16	2,9	2,42
30	12,2	15,7	14,8	3,39	4,13	3,99	4,09	3,63	4,04	3,28	3,16

- **Evaluación normativa**

La prueba de rendimiento TCP se valida conforme a la RFC 6349, la cual proporciona un marco normativo para la evaluación de throughput en redes IP modernas. Esta RFC recomienda el uso de herramientas como `iperf` para realizar pruebas activas sobre TCP, considerando parámetros como el tamaño de ventana, latencia y condiciones reales de congestión. La Ec.2 permite calcular el throughput TCP en caso de no obtenerlo de manera explícita al ejecutar el comando descrito en la Tabla 13:

$$\textbf{\textit{Throughput}}\ (\textbf{\textit{Mbps}}) = \frac{\textbf{\textit{Tamaño de la ventana de recepción TCP}} * \mathbf{8}}{\textbf{\textit{Tiempo de ida y vuelta}}\ (\textbf{\textit{RTT}})} \qquad \textit{(Ec.2)}$$

En este experimento, los valores de throughput obtenidos mediante la simulación y presentados en la **Tabla 14**, son consistentes con lo esperado en entornos SDN con VXLAN, donde los flujos intra-VTEP muestran mayor eficiencia que los flujos inter-VTEP debido a la menor complejidad en el plano de datos.

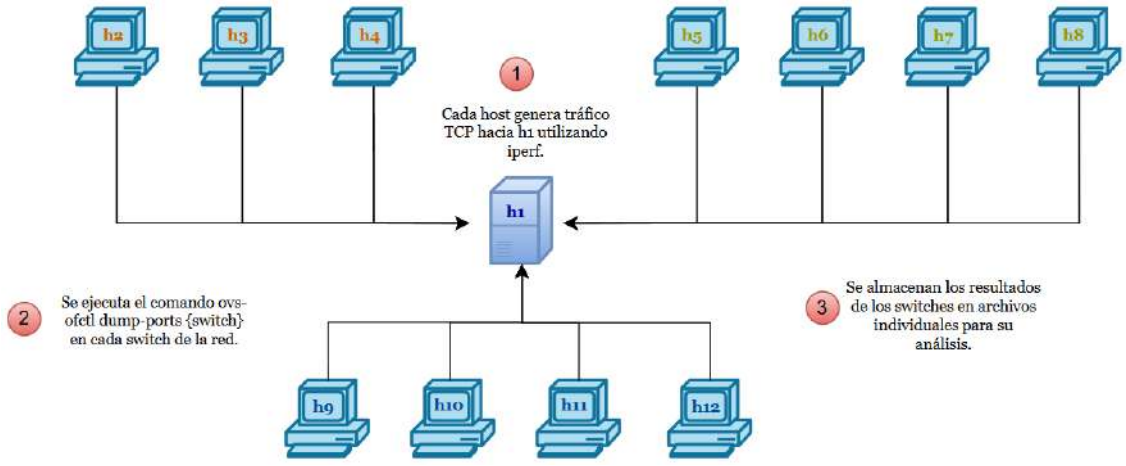
Se concluye que la red cumple con el marco definido por RFC 6349, ya que la metodología utilizada se alinea con las buenas prácticas de medición TCP. No obstante, los resultados también revelan oportunidades de mejora en el balanceo de carga y en la optimización del tránsito entre VTEPs.

4.3.4 PRUEBA 4: Análisis del impacto del balanceo de carga

El objetivo de la prueba es analizar la distribución del tráfico en la red y evaluar si los dispositivos switches se encuentran gestionando eficientemente el balanceo de carga entre los flujos. En una infraestructura SDN-VXLAN, la distribución equitativa del tráfico es importante para evitar congestión en enlaces específicos y garantizar un uso óptimo de los recursos de red. Un balanceo de carga eficiente permite la mejora del rendimiento y la estabilidad de la red, minimizando aspectos como la latencia y maximizando el throughput. A continuación, la **Figura 54**, describe la metodología que se utiliza para la ejecución de esta prueba.

Figura 54

Metodología para la simulación de la prueba 4.



La **Tabla 14** demuestra cada parámetro del comando `<host_cliente> iperf -c 10.0.0.1 -p 5001 -t 30` que se utiliza para ejecutar una prueba de balance de carga en una red, utilizando iperf como herramienta. Estas pruebas permiten evaluar cómo se distribuye

el tráfico entre diferentes servidores, asegurando un uso eficiente de los recursos de red y evitando la sobrecarga en un solo punto.

Tabla 14

Descripción de los parámetros para ejecución de prueba de balance de carga.

Comando	Función
(h2, h3, h4, h5…h12)	Host cliente
iperf	Ejecuta la herramienta iperf (versión 2)
-c <IP_SERVIDOR>	Indica que es cliente y se conecta a la IP del servidor
-p 5001	Usa puerto por defecto de iperf versión 2
-t 30	Ejecuta la prueba durante 30 segundos

La **Figuras 55, 56 y 57**, especifican las estadísticas relacionadas con el tráfico de red para cada puerto del dispositivo conmutador (VTEP), haciendo referencia a la cantidad de datos transmitidos o recibidos en cada puerto de manera específica o concreta. El objetivo de estas estadísticas es proporcionar los datos (bytes) que nos permitirán visualizar, de manera conjunta, la interacción del tráfico. Cabe destacar que el enfoque de esta prueba se centra en los equipos s3, s4 y s5 de la topología.

Figura 55

Evidencia de las pruebas de balanceo en el txt para s3.

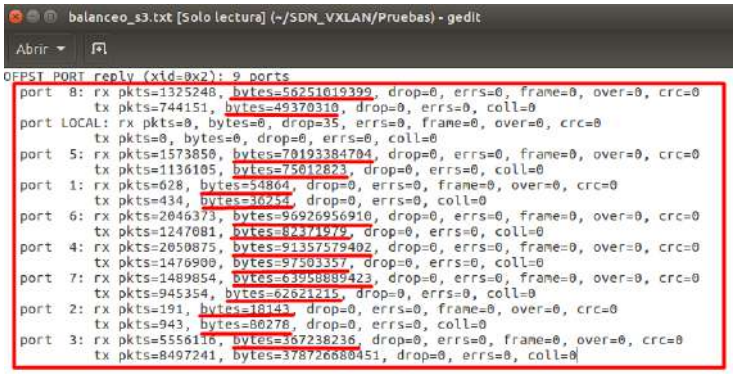


Figura 56

Evidencia de las pruebas de balanceo en el txt para s4.

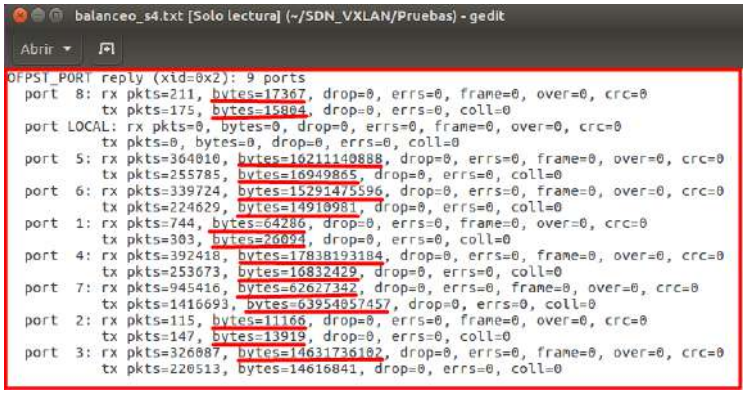
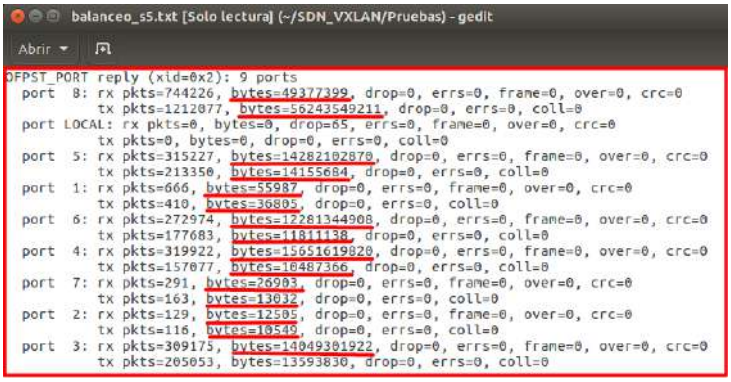


Figura 57

Evidencia de las pruebas de balanceo en el txt para s5.



- **Análisis en base a la ejecución de la prueba**

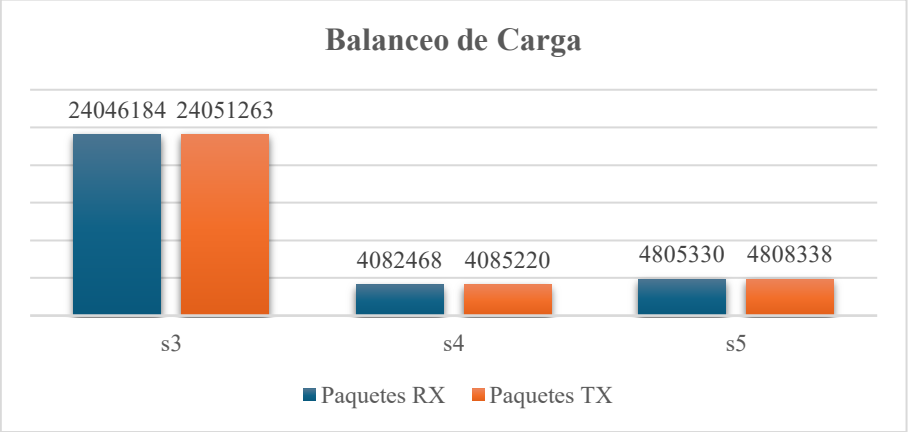
Los resultados obtenidos presentados en las **Figuras 52, 53 y 54,** correspondientes a s3, s4 y s5 respectivamente, muestran que el switch s3 concentró más del 75% del total de paquetes, mientras que s4 y s5 manejaron menos del 25% en conjunto. Esta diferencia responde a un hecho topológico importante: el host h1 (destino del tráfico) está conectado físicamente al switch s3, por lo que todo el tráfico generado por los emisores termina atravesando este nodo.

Por tanto, el desbalance observado no implica una falla de diseño, sino una consecuencia directa del rol central de s3 como punto de agregación final de tráfico hacia h1.

Es importante entender que, para el análisis, se toman los valores de cada puerto, tanto de TX como de RX, para realizar una suma que nos mostrará la cantidad total de datos transferidos entre los equipos, como se demuestra en la **Figura 58.**

Figura 58

Resultados de Balanceo de Carga total para cada equipo.



- **Evaluación normativa**

La recomendación **ITU-T Y.1540**, en su Sección 6.11, sugiere que el análisis del rendimiento IP debe considerar posibles puntos de concentración de tráfico que puedan comprometer el desempeño del sistema. Aunque esta recomendación advierte sobre cuellos de botella, también establece que dichos puntos deben evaluarse en el contexto de la arquitectura implementada.

Para determinar el nivel de participación relativa de cada switch, se utiliza la Ec.3:

$$\%_{tráfico\ en\ s3} = \frac{\boldsymbol{Paquetes_{RXs3}}}{\boldsymbol{Total\ de\ paquetes\ RX}} * \mathbf{100} \qquad \textit{(Ec.3)}$$

$$\%_{tráfico\ en\ s3} = \frac{\mathbf{24046184}}{\mathbf{24046184 + 4082468 + 4805330}} * \mathbf{100 = 75.1\%}$$

Dicho resultado refleja que s3 (VTEP 1) es el principal punto de concentración de tráfico, lo cual está justificado debido a la conexión directa con el servidor h1. Los switches s4 (VTEP 2) y s5 (VTEP 3) funcionaron correctamente como puntos de tránsito intermedios, pero no como nodos de terminación.

La infraestructura de red indica un comportamiento coherente con su topología, en la que el tráfico naturalmente converge en s3 (VTEP 1) por ser el nodo de destino. Aunque la carga no está equilibrada entre todos los switches, esto no representa una condición de congestión ni incumplimiento normativo, pues no se evidenciaron pérdidas ni colapsos de rendimiento durante la prueba. En base a ITU-T Y.1540, el sistema se considera funcional, aunque pueden explorarse mejoras en el plano de reenvío para balancear rutas si se diversifica el destino de los flujos.

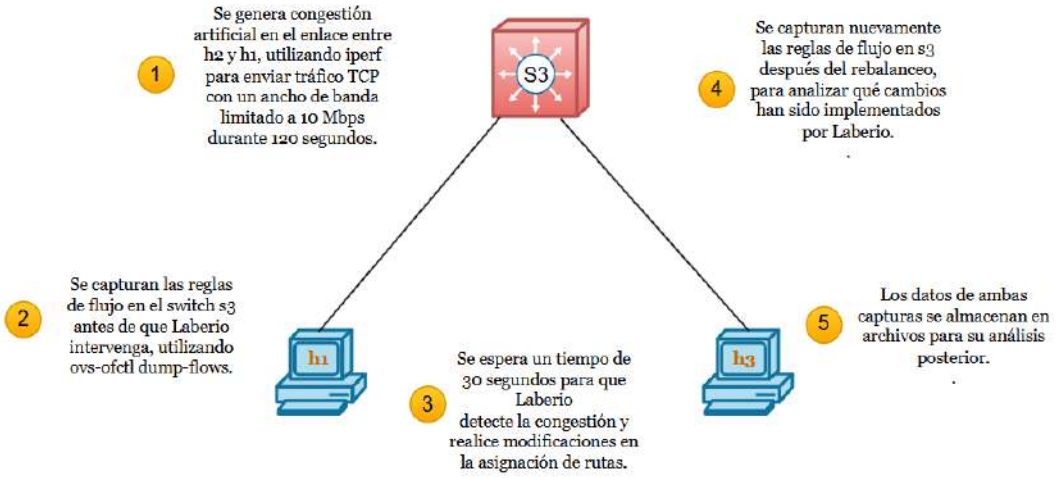
4.3.5 PRUEBA 5: Simulación de Congestión y Análisis de Laberio

La prueba tiene como propósito la evaluación del comportamiento de la infraestructura de red ante acción de congestión en un enlace específico y analizar la respuesta del mecanismo empleado para el balanceo de carga y la adaptación de flujos gestionada por Laberio.

Laberio en sí, es un framework que optimiza el tráfico en redes SDN, ajustando dinámicamente las reglas de flujo para mitigar la congestión, mejorar la distribución del tráfico y optimizar el encaminamiento de paquetes dentro de la red. Esta prueba busca determinar si, tras la intervención de Laberio, la asignación de rutas cambia de manera efectiva y si la red logra una distribución más equitativa de la carga, garantizando así un uso más eficiente de los recursos disponibles en la red. A continuación, la **Figura 59**, presenta la metodología que se utiliza para la ejecución de esta prueba.

Figura 59

Metodología para simulación de prueba 5.



La **Tabla 15** presenta los parámetros del comando `h2 iperf -c 10.0.0.1 -p 5001 -t 30 -b 10M &`, utilizados para realizar pruebas de congestión de enlace mediante la herramienta iperf. Esta acción permite evaluar el rendimiento de la red al medir el tráfico y el ancho de banda disponible entre un cliente y un servidor.

Tabla 15

Descripción de parámetros para prueba de congestión de enlace.

Comando	Función
h2	Host desde el que se ejecuta el comando (cliente)
iperf	Ejecuta la herramienta iperf
-c <IP_SERVIDOR>	Indica que es cliente y se conecta a la IP del servidor
-p 5001	Usa puerto por defecto de iperf
-t 30	Ejecuta la prueba durante 30 segundos
-b 10M	Establece un ancho de banda de 10 Mbps (solo en modo UDP)

&	Ejecuta el comando en segundo plano

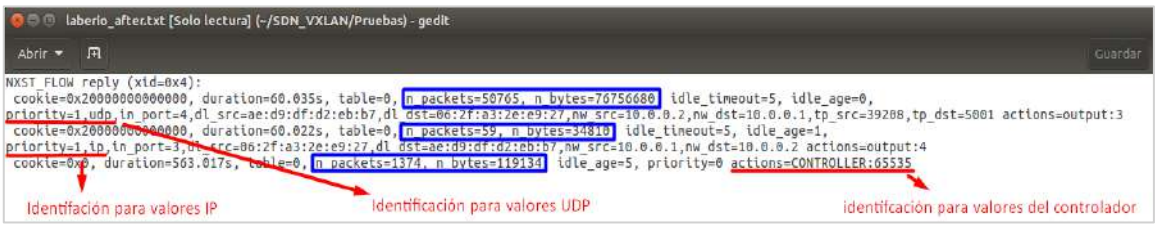
En las **Figuras 60 y 61** se presentan los archivos de texto que contienen la información sobre las estadísticas de flujo de red obtenidas. El enfoque establecido para esta prueba se centra en aspectos como el número de bytes y el número de paquetes, tanto para UDP, IP como para el controlador.

Figura 60

Evidencia de la captura antes de la congestión.

Figura 61

Evidencia de la captura después de la congestión.



- **Análisis en base a la ejecución de la prueba**

Los resultados obtenidos en la Prueba 5, mostrados en las **Figuras 60 y 61** evidencian que la intervención del framework Laberio tuvo un impacto positivo en la optimización del tráfico de red bajo condiciones de congestión. Se observó un incremento significativo en la cantidad de paquetes UDP, pasando de 25237 a 50735, y en el volumen de datos UDP, que se duplicó de 38158344 a 76711320 bytes. De igual manera, los paquetes y bytes IP también incrementaron considerablemente, indicando una mejora en la capacidad de transmisión y en la

eficiencia del manejo del tráfico a través de los switches. Este resultado sugiere que Laberio permite la redistribución más efectiva de la carga, reduciendo la congestión sin generar pérdida de rendimiento.

Por otra parte, los datos como el número de paquetes enviados al controlador SDN demuestran una variación mínima (de 1357 a 899), evidenciando que la optimización aplicada no sobrecargó el plano de control. Este comportamiento es esperable en entornos SDN, pues mantiene la estabilidad de la red al mismo tiempo que mejora la utilización de los recursos, además de no necesitar una gestión constante por parte del controlador. En conjunto, los datos reflejan que Laberio es una herramienta efectiva para gestionar congestión en arquitecturas SDN, permitiendo mejorar el rendimiento global sin comprometer la arquitectura del sistema.

La **Figura 62** ilustra de manear clara los cambios, demostrando que Laberio logra mejorar el rendimiento general del tráfico de red, optimizando la gestión de paquetes.

Figura 62

Resultados de prueba de congestión en enlace.

- **Evaluación normativa**

La recomendación **ITU-T Y.1540,** en su Sección 6.4, establece que las redes IP deben ser evaluadas en condiciones de operación degradada (como congestión) y que deben demostrar capacidad de recuperación, redistribución de tráfico y mantenimiento de estabilidad.

Para validar matemáticamente la mejora de manera cuantitativa, se utiliza la Ec.4 de incremento porcentual:

$$\boldsymbol{Incremento}\ (\%) = \frac{\boldsymbol{Despues - Antes}}{\boldsymbol{Antes}} \times \mathbf{100} \qquad \textit{(Ec.4)}$$

Aplicando Ec.4, se obtuvieron los valores porcentuales que se detallan a continuación en la **Tabla 16**:

Tabla 16

Resultados obtenidos del incremento porcentual mejorada.

Variable	Antes	Después	Incremento (%)
n_packets UDP	25237	50735	101,04
n_bytes UDP	38158344	76711320	101,04
n_packets IP	29	59	103,45
n_bytes IP	17110	34810	103,48
n_packets Controller	1357	899	-33,77
n_bytes Controller	117888	149913	27,17

- **UDP/IP** demuestra un aumento considerable (aproximadamente un 101 % y 103 % en el ejemplo indicado), lo que representa una mejora sustancial en la eficiencia de transmisión. Este incremento ayuda a sostener que el volumen de paquetes y bytes transmitidos tanto en por el protocolo UDP como en el de IP se duplica, evidenciando un notable crecimiento en la capacidad o rendimiento de la red.

- El tráfico al controlador presenta comportamientos mixtos: mientras el número de paquetes baja en aproximadamente un 33.8%, el volumen de bytes aumentó en un 27.2%. Esto sugiere una variación en el tipo o tamaño del tráfico hacia el controlador, pero en términos generales no afecta negativamente la gestión del plano de control.

En conclusión, se puede afirmar que el framework Laberio cumple con los criterios establecidos por la normativa ITU-T Y.1540 para redes en condiciones de congestión. Mejora significativamente la capacidad de transmisión sin comprometer el plano de control ni causar pérdida de paquetes. Esta optimización mantiene la eficiencia y estabilidad del sistema SDN-VXLAN en escenarios reales con tráfico variable o intensivo.

En general, la realización de estas pruebas es esencial para garantizar la eficiencia, estabilidad y predictibilidad de una red, especialmente en entornos que utilizan tecnologías como VXLAN y SDN, que permiten la virtualización y segmentación del tráfico. Estas evaluaciones son clave para verificar que los mecanismos de encapsulamiento, aislamiento de dominios de broadcast y control centralizado funcionen correctamente, incluso cuando hay variaciones en la carga o cambios en la topología.

CONCLUSIONES

La virtualización de redes mediante el protocolo VXLAN, combinada con el control centralizado de la arquitectura de red SDN, permiten separar la topología lógica de la física. Esta arquitectura facilita la segmentación del tráfico por medio de VNIs (Virtual Network Identifiers), lo que aporta mayor flexibilidad en la administración de servicios de red, especialmente en entornos dinámicos y de gran escala. Además, permite un mejor aprovechamiento de los recursos disponibles y una gestión ágil frente a cambios en la infraestructura.

La revisión teórica que se realiza permite identificar los fundamentos técnicos y las tendencias actuales en el desarrollo de redes virtuales. Esto no solo facilita la selección adecuada de herramientas y metodologías para el diseño e implementación de la infraestructura, sino que también permite contextualizar los beneficios de SDN y VXLAN dentro del marco de las necesidades modernas de escalabilidad, automatización y eficiencia operativa.

Utilizando entornos virtuales como Mininet y controladores como Floodlight, se logra simular y poner en funcionamiento una red que integra el plano de control definido por software y la encapsulación VXLAN. La demostración práctica evidencia que es posible coordinar el enrutamiento de flujos a través de túneles virtuales definidos dinámicamente, reduciendo la dependencia de configuraciones manuales y mejorando la eficiencia operativa de la red.

El uso de una metodología de desarrollo en cascada es esencial para estructurar el proyecto desde la fase de análisis hasta la evaluación de resultados. Cada etapa permite cumplir objetivos específicos, minimizando errores y facilitando el seguimiento del avance. Esta organización metodológica también hizo posible realizar ajustes durante la implementación y evaluación, lo cual es de importancia en el desarrollo de infraestructuras tecnológicas.

A través de pruebas de conectividad, rendimiento, latencia, congestión y balanceo de carga, se comprueba que la infraestructura propuesta mejora la entrega de servicios y responde adecuadamente a situaciones variables en la red. La gestión centralizada, junto con la capacidad de segmentar el tráfico y crear redes virtuales independientes, se traduce en una mayor eficiencia y escalabilidad, confirmando que esta combinación tecnológica representa una alternativa robusta frente a las arquitecturas de red tradicionales.

Desde una perspectiva administrativa, la implementación de una arquitectura virtualizada basada en SDN y VXLAN sobre una topología Spine-Leaf simplifica considerablemente la gestión y configuración de la red. Al desacoplar el plano de control del plano de datos y permitir la creación dinámica de redes superpuestas mediante VNIs, se reduce la dependencia del hardware físico tradicional y se habilita un control centralizado, programable y altamente flexible. Esta capacidad resulta especialmente valiosa en entornos que requieren alta escalabilidad, rápida adaptabilidad y segmentación segura del tráfico. Además, la estructura Spine-Leaf asegura múltiples rutas y un balanceo de carga más eficiente, lo que mejora el rendimiento de la infraestructura cuando se gestiona mediante SDN.

RECOMENDACIONES

Para optimizar la integración entre SDN y VXLAN en futuras investigaciones y aplicaciones prácticas, se recomienda explorar mejoras en la eficiencia del tráfico de red mediante la optimización de los algoritmos de enrutamiento y balanceo de carga en entornos altamente escalables. La combinación de estas tecnologías ya ha demostrado beneficios en términos de segmentación y administración centralizada, pero aún pueden analizarse estrategias para reducir la sobrecarga en el procesamiento de paquetes y minimizar la latencia en escenarios con un alto volumen de tráfico.

Otro aspecto clave a considerar es la implementación de mecanismos avanzados de seguridad en la interconexión de redes virtuales. Aunque VXLAN permite la segmentación del tráfico mediante VNIs, es fundamental integrar soluciones de cifrado y autenticación en los túneles VXLAN para prevenir vulnerabilidades, especialmente en infraestructuras distribuidas. Incorporar políticas de seguridad basadas en SDN permitiría gestionar de manera dinámica y adaptable las reglas de acceso, reforzando la protección de la red contra ataques o accesos no autorizados.

En cuanto al rendimiento, es recomendable evaluar el impacto de diferentes controladores SDN en la eficiencia de la red, ya que cada implementación ofrece características y optimizaciones específicas. Comparar soluciones como ONOS, OpenDaylight y Floodlight podría proporcionar una visión más completa sobre cuál es la más adecuada según el tipo de aplicación e infraestructura en la que se despliega la red. Además, analizar el uso de protocolos emergentes como BGP EVPN para la distribución de información sobre VXLAN podría mejorar aún más la escalabilidad y flexibilidad de la red.

Desde una perspectiva práctica, sería útil realizar pruebas en escenarios híbridos donde coexistan infraestructuras tradicionales con redes basadas en SDN y VXLAN. Este enfoque

permitiría evaluar la viabilidad de una transición progresiva hacia estas tecnologías sin afectar la operatividad de la red. También se recomienda explorar la automatización de la configuración y gestión de la red mediante otras herramientas.

Finalmente, para maximizar los beneficios de SDN y VXLAN en entornos empresariales y de centros de datos, se recomienda explorar la integración con tecnologías emergentes como la inteligencia artificial y el aprendizaje automático. Implementar soluciones que automaticen la optimización del tráfico y la detección de anomalías en la red permitiría mejorar la eficiencia operativa y la capacidad de respuesta ante cambios en la demanda o incidentes de seguridad.

BIBLIOGRAFÍA

Acosta, S., & Ortega, E. (2023). *ANÁLISIS SISTEMÁTICO DE MODELOS DE SEGURIDAD Y CONTROL DE ACCESO EN ARQUITECTURA SDN PARA LA DETENCIÓN DE ANOMALÍAS*.

Aguirre F., & Crespo M. (2021). *CONVERGENCIA DE REDES DEFINIDAS POR SOFTWARE CON OPENDAYLIGHT Y NSX SOBRE UNA NUBE PRIVADA*.

Amaya Fariño, L. M., Arroyo Pizarro, J. F., Jaramillo Infante, M., Tumbaco Reyes, A. R., & Mendoza Morán, B. M. (2022). SDN Redes definidas por Software usando MiniNet. *Revista Científica y Tecnológica UPSE*, *9*(1), 48–56. https://doi.org/10.26423/rctu.v9i1.489

Castro Serantes, B. (2023). *ESTUDIO E IMPLEMENTACIÓN DE TECNOLOGÍAS OVERLAY EN REDES DEFINIDAS POR SOFTWARE SDN*.

Cobos Yilder. (2021). *DISEÑO E IMPLEMENTACIÓN DE UNA RED VIRTUAL BASADA EN DOCKER EN UN AMBIENTE DE REDES DEFINIDAS POR SOFTWARE (SDN). UTILIZANDO ZEROTIER Y RASPBERRY PI".*

Dutt, D., Networks, C., Duda, K., Agarwal, A. P., Kreeger, B. L., Sridhar, C. T., Bursell, V. M., & Wright, I. C. (2014). *Independent Submission M. Mahalingam Request for Comments: 7348 Storvisor Category: Informational Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. http://www.rfc-editor.org/info/rfc7348.

Huawei. (2022). *What Is VXLAN - Huawei*. https://support.huawei.com/enterprise/en/doc/EDOC1100086966

Leonardo, J., Ramirez, H., Tecnologica, U., Pereira, D. E., & De Ingenierias, F. (2015). *GUÍA DE IMPLEMENTACIÓN Y USO DEL EMULADOR DE REDES MININET*.

Manuel, C., Vergel, R., & Calderón, C. A. (2014). *Controladores SDN, elementos para su selección y evaluación*. https://www.researchgate.net/publication/320711755

Mendiola, A., Astorga, J., Jacob, E., & Higuero, M. (2017). A Survey on the Contributions of Software-Defined Networking to Traffic Engineering. In *IEEE Communications Surveys and Tutorials* (Vol. 19, Issue 2, pp. 918–953). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/COMST.2016.2633579

Mininet Project Contributors. (2022). *Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet*. https://mininet.org/

Nadeau, T. D., & Gray, K. (2013). *SDN: Software Defined Networks*.

Naranjo E, & Salazar G. (2017). *Underlay and Overlay Networks: The approach to solve addressing and segmentation problems in the new networking era*.

Norberto Figuerola. (2014). *SDN-Redes definidas por Software*.

Open Networking Foundation. (2024). *Floodlight. GitHub.* Https://Github.Com/Floodlight/Floodlight.

Paucar A. (2022). *ANÁLISIS E IMPLEMENTACIÓN DE UN PROTOTIPO DE RED SDN EN EL CAMPUS NORTE DE LA UNACH*.

Pérez E, & Molina K. (2023). *Diseño e Implementación de VLANs en Entornos Universitarios Basados en Redes Definidas por Software (SDN)*.

RFC7348. (2014). http://www.rfc-editor.org/info/rfc7348.

Rodríguez Herlein, D. R., Talay, C. A., González, C. N., & Marrone, L. A. (2020). *Explorando las redes definidas por software (SDN)*.

Ruano, F. L., Luis Muñoz, J., & Barcelona, T. (2017). *CREATION OF A VIRTUAL OVERLAY NETWORK WITH SDN AND VXLAN*.

Salazar, G. D. (2021). *Hybrid Networking SDN y SD-WAN: Interoperabilidad de Arquitecturas de Redes Tradicionales y Redes definidas por Software en la era de la digitalización*.

Stallings, W. (2015). *Foundations of Modern Networking SDN, NFV, QoE, IoT, and Cloud*. Indiana: Pearson Education, Inc.

Underdahl Brian, & Kinghorn Gary. (2015). *Software Defined Networking*.

VMWare. (2023). *¿Qué son las redes definidas por software (SDN)? | Glosario de VMware | ES*. https://www.vmware.com/es/topics/glossary/content/software-defined-networking.html

Ynga, S., Asesor, R., Ing, M. M., & Cabrera, I. (2020). *FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS ARQUITECTURAS DE REDES DE COMPUTADORAS DEFINIDAS POR SOFTWARE: REVISIÓN BIBLIOGRÁFICA*.

ANEXO

Instalación y configuración de las herramientas para el diseño.

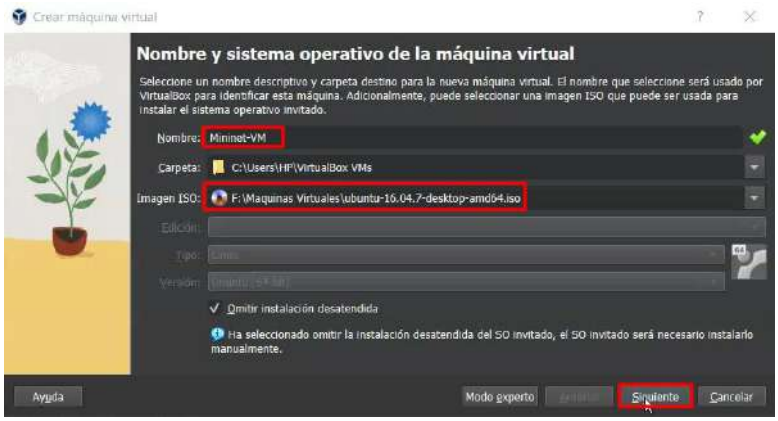
En esta sección se detallan los procedimientos necesarios para la instalación y configuración de las herramientas empleadas en el diseño de redes que integran tecnologías SDN y VXLAN. Entre los componentes clave se incluyen el sistema operativo base, el emulador Mininet y el controlador SDN Floodlight. Además, se describe cómo preparar estos elementos dentro de una máquina virtual, con el fin de garantizar su correcta operación y compatibilidad durante las pruebas de conectividad y funcionalidad en el entorno de red.

Instalación del Sistema Operativo

En la **Figura 63** se ilustra el procedimiento para crear una máquina virtual destinada a la instalación de Mininet. Durante este proceso, se asigna un nombre identificativo, como "Mininet-VM", y se define la ubicación en el sistema donde se almacenarán los archivos asociados a dicha máquina virtual. Además, se seleccionada una imagen ISO de Ubuntu 16.04.7, que será el sistema operativo base sobre el cual se instalará la herramienta Mininet. Esta versión de Ubuntu es compatible y recomendada para simular redes con Mininet y sobre todo el controlador Floodlight.

Figura 63

Configuración del nombre y sistema operativo de la VM.

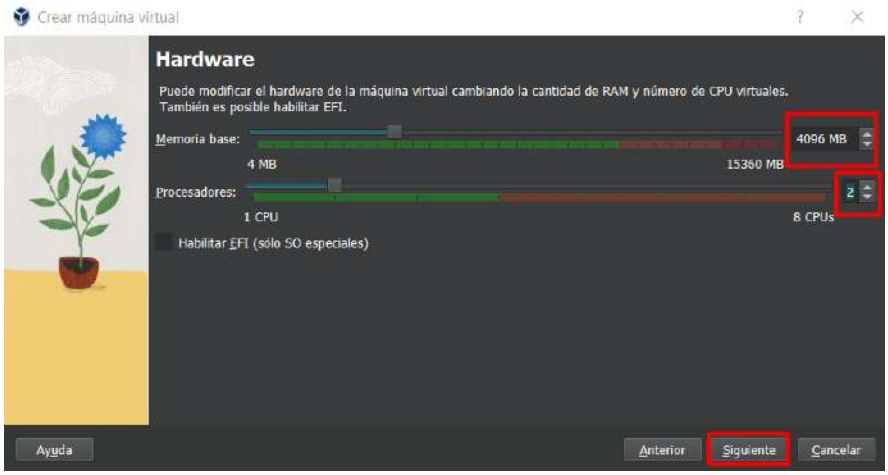


Para la configuración de hardware de la VM que se presenta en la **Figura 64,** se asignan 4096 MB (4 GB) de memoria RAM, un valor considerable para garantizar que el sistema operativo Ubuntu, junto con Mininet, puedan funcionar de manera fluida y precisa. Esta cantidad de memoria es suficiente para manejar tareas relacionadas con la simulación de redes, asegurando un buen rendimiento sin comprometer los recursos del equipo anfitrión o algún otro inconveniente.

Además, se asignan dos de los ocho procesadores disponibles en el sistema físico a la máquina virtual, lo que le otorga una capacidad de cómputo adecuada para ejecutar simulaciones exigentes en Mininet. Esta configuración contribuye a una distribución eficiente de la carga de trabajo, permitiendo que la máquina virtual realice múltiples tareas al mismo tiempo sin experimentar disminuciones en el rendimiento.

Figura 64

Configuración del hardware de la máquina virtual.

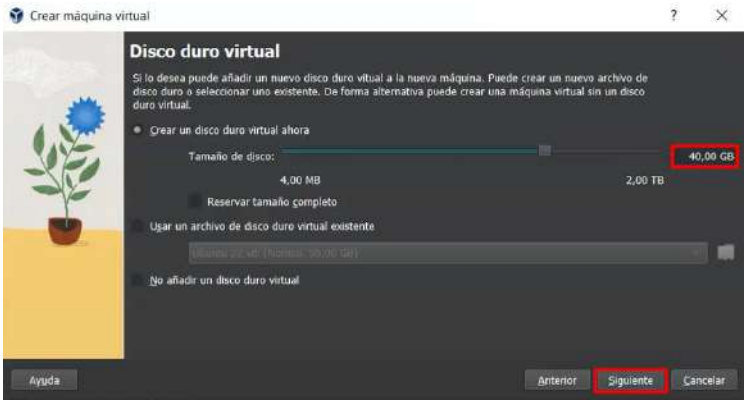


Para el próximo paso como se muestra en la Figura 65, se configura la creación de un disco duro virtual para la VM, asignando 40 GB de almacenamiento. Dicha cantidad es adecuada para instalar Ubuntu y Mininet, además de permitir espacio suficiente para almacenar archivos y simulaciones que se generen durante su uso. La opción de no reservar el tamaño

completo optimiza el uso del almacenamiento en el equipo anfitrión, permitiendo que el espacio se asigne según sea necesario. Con esta configuración, se garantiza que la máquina virtual tenga un entorno adecuado para su correcto funcionamiento.

Figura 65

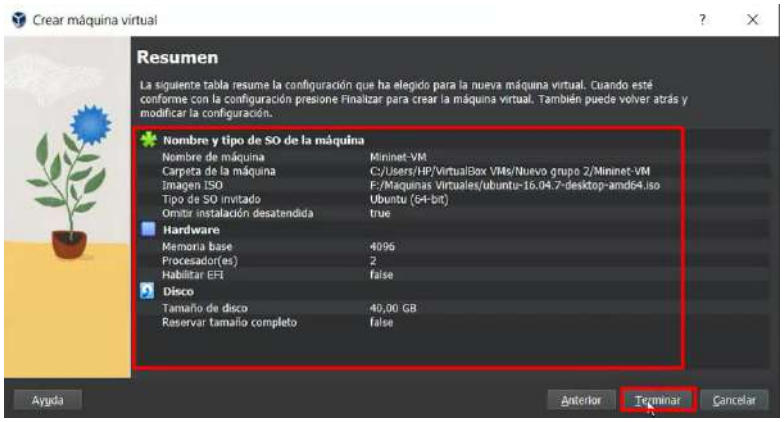
Creación del disco duro virtual.



En el resumen final que se presenta a la **Figura 66** se puede observar la configuración de la máquina virtual, se confirma que se creará bajo el nombre "Mininet-VM" con un sistema operativo Ubuntu 16.04.7 de 64 bits. Se le asignarán 4096 MB de RAM, 2 procesadores y un disco duro virtual de 40 GB, sin reservar el tamaño completo, optimizando así el uso del almacenamiento del equipo anfitrión

Figura 66

Resumen de la configuración de la máquina virtual.



Una vez establecidas las características y capacidades principales para la VM, se procede a iniciar la instalación del sistema operativo seleccionando el tipo de instalación que requeriremos, como se presenta en el ejemplo de la **Figura 67.**

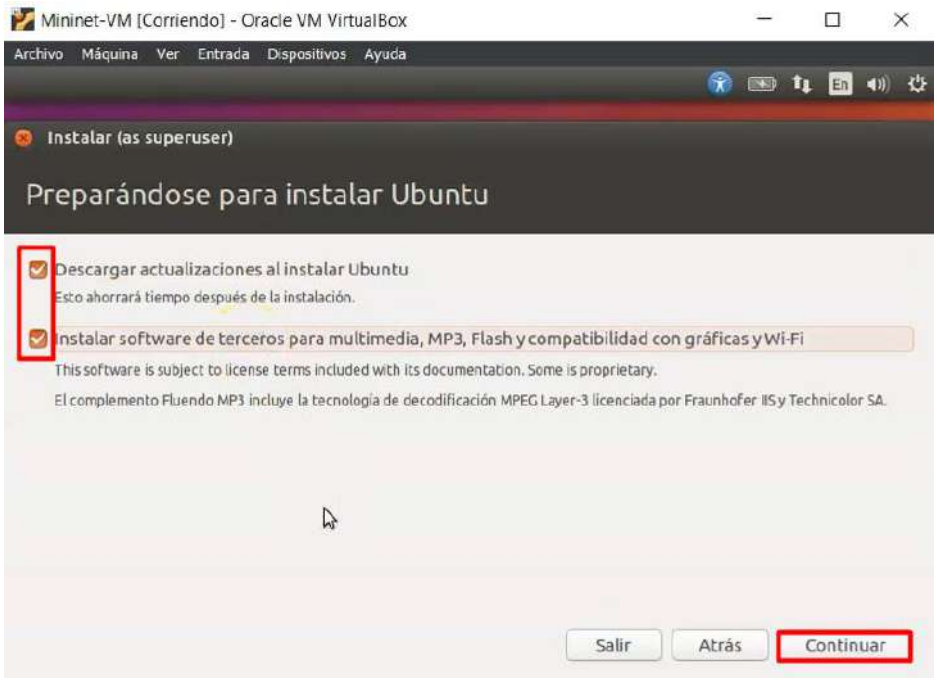
Figura 67

Pantalla de bienvenida de la instalación de Ubuntu.

Durante la instalación de Ubuntu, se seleccionan dos opciones importantes, como se muestra en la **Figura 68**. La primera consiste en permitir que el sistema descargue actualizaciones mientras se instala, lo cual resulta útil para ahorrar tiempo, ya que evita realizar estas actualizaciones manualmente después. La segunda opción activa la instalación de software de terceros, incluyendo códecs para reproducir archivos multimedia como MP3, soporte para gráficos propietarios, Wi-Fi y Flash. Esto garantiza una mejor compatibilidad del sistema desde el inicio. Al continuar con el proceso, Ubuntu se instala con todos estos componentes preconfigurados, lo que facilita su uso inmediato dentro de la máquina virtual.

Figura 68

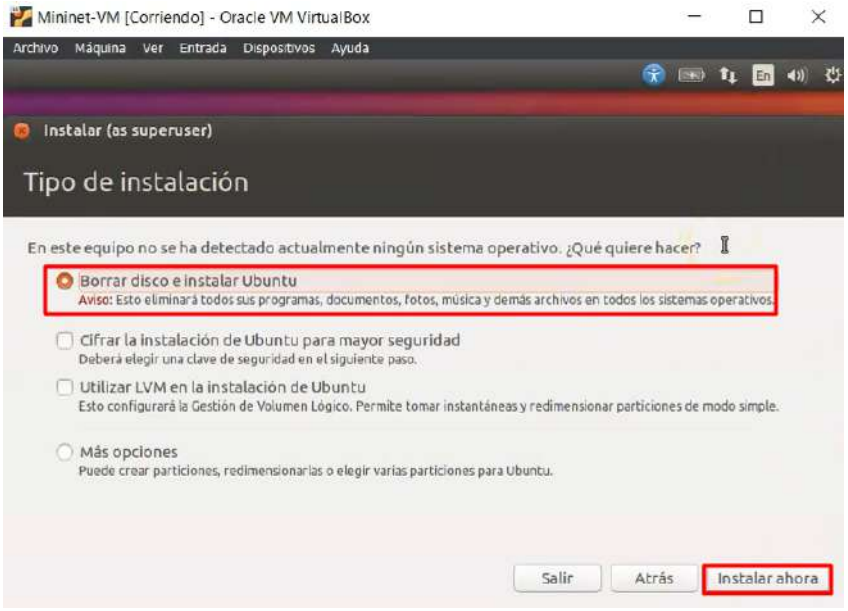
Preparación para la instalación de Ubuntu.



En la **Figura 69** se muestra la selección del tipo de instalación de Ubuntu. En este caso, se opta por la opción “Borrar disco e instalar Ubuntu”, lo que significa que el disco virtual asignado a la máquina virtual será formateado por completo y se instalará el sistema operativo desde cero. Aunque esta acción elimina cualquier dato previo, no representa un riesgo, ya que se trata de una VM nueva sin sistemas operativos o archivos anteriores. Esta decisión permite comenzar con un entorno limpio, lo cual es fundamental para asegurar que la instalación se realice sin conflictos y el sistema funcione de manera óptima. Al hacer clic en “Instalar ahora”, se inicia el proceso de instalación de Ubuntu en la máquina virtual.

Figura 69

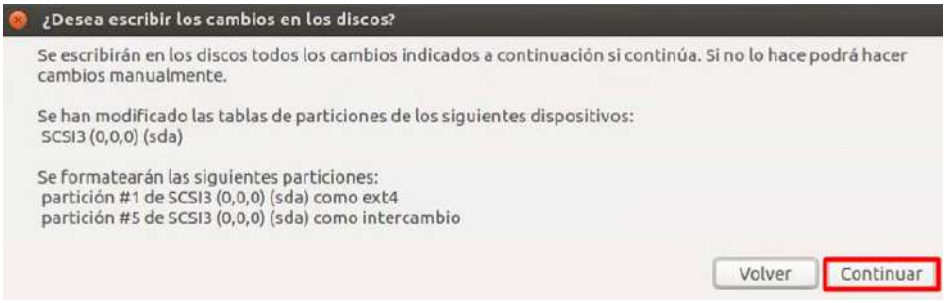
Selección del tipo de instalación de Ubuntu.



El sistema confirma los cambios que se harán en el disco virtual antes de proceder con la instalación de Ubuntu. Se informa que las tablas de particiones han sido modificadas en el dispositivo, lo que incluye la creación de dos particiones: la partición #1, que será formateada en ext4, el sistema de archivos predeterminado para Linux, y la partición #5, que se utilizará como partición de intercambio (swap), destinada a mejorar la gestión de la memoria en el sistema como se aprecia en la **Figura 70.**

Figura 70

Confirmación de los cambios en el disco.



Como configuración previa el instalador de Ubuntu solicita el seleccionamiento de la zona horaria del sistema operativo. En este caso, se ha seleccionado la ciudad de Guayaquil, que corresponde a la zona horaria de Ecuador como en la **Figura 71**. Esto garantiza que la máquina virtual tenga la hora local correcta, lo cual es importante para la sincronización de tareas y el correcto funcionamiento de las aplicaciones que dependen del tiempo.

Figura 71

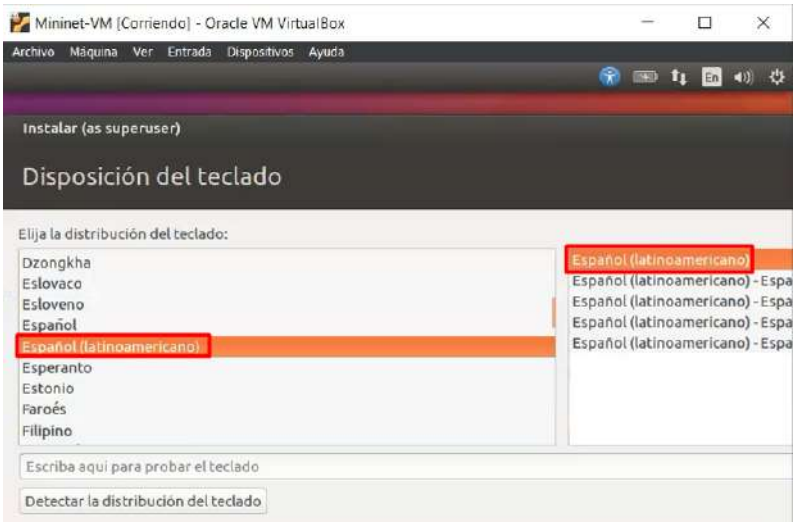
Configuración de la zona horaria.

Otro paso importante es que el instalador de Ubuntu solicita seleccionar la disposición del teclado que se utilizará en el sistema, que para una mejor práctica se selecciona el "español (latinoamericano)", lo que asegura que el teclado funcionará de acuerdo con la distribución comúnmente utilizada en América Latina, permitiendo que los caracteres especiales, como la "ñ", estén correctamente mapeados en el teclado como en se muestra en la **Figura 72.**

Definir correctamente la distribución del teclado es un paso clave para asegurar una experiencia de uso fluida al interactuar con el sistema operativo. Esta configuración influye directamente en la forma en que se escriben comandos, contraseñas y cualquier otro tipo de texto, por lo que es fundamental que coincida con el idioma y el formato del teclado físico utilizado.

Figura 72

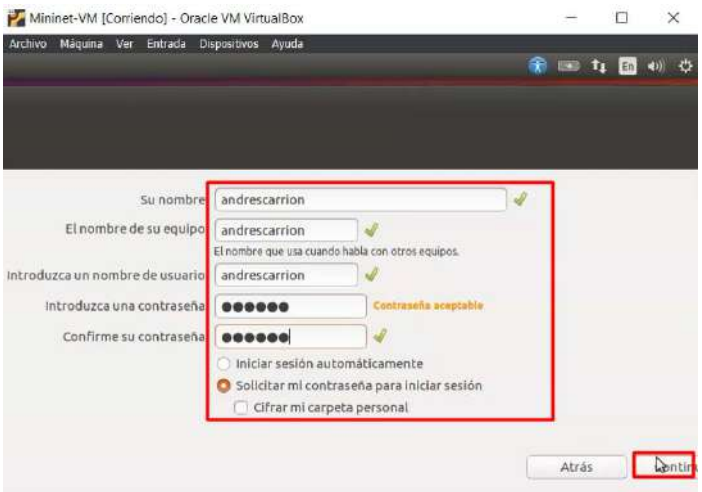
Configuración de la disposición del teclado.



Finalmente, se establece la cuenta principal que administrará el sistema operativo y garantizar que el acceso esté protegido por una contraseña segura como en la **Figura 73**. Se reinicia la máquina para que se realicen las configuraciones establecidas para el posterior uso y listo.

Figura 73

Configuración del usuario principal.

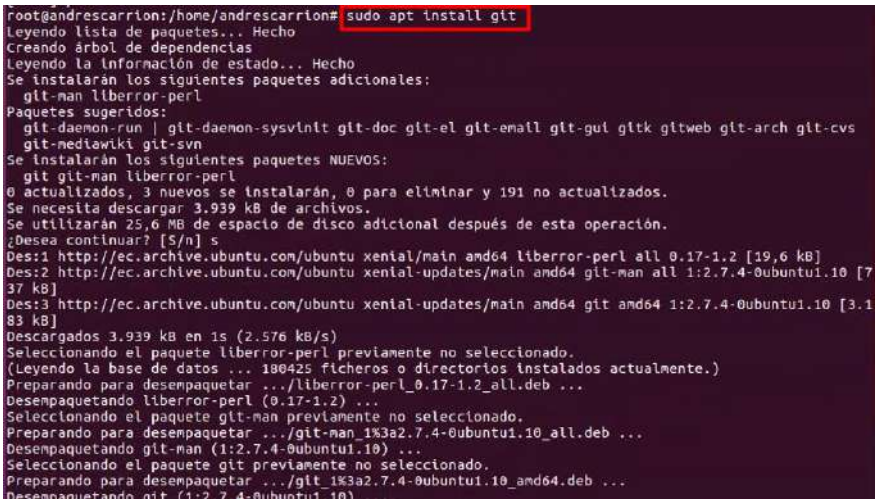


Instalación del controlador

El proceso de instalación requiere varias herramientas auxiliares, entre los primeros pasos para la implementación de un controlador SDN como Floodlight es la instalación de Git, un sistema de control de versiones distribuido que permite acceder al repositorio de desarrollo de Floodlight. Git es esencial no solo para la descarga del código fuente, sino también para mantener el software actualizado con los últimos cambios en el repositorio, algo crucial en un proyecto de tecnologías emergentes como SDN. A continuación, se expone de manera práctica lo mencionado en la **Figura 74.**

Figura 74

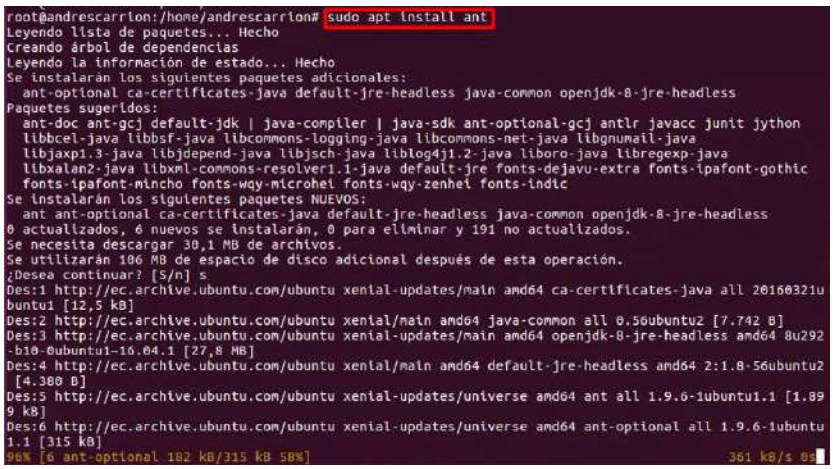
Instalación de herramienta Git en el sistema.



En la **Figura 75** se muestra que para la instalación y correcta ejecución del controlador SDN Floodlight, es necesario contar con `Apache Ant`, una herramienta de automatización de compilación para proyectos de software, mediante el comando `sudo apt install ant`. Al ejecutar el comando correspondiente, Ubuntu se encarga automáticamente de descargar e instalar todas las dependencias necesarias para el funcionamiento de Ant. Entre estas se incluyen el Java Development Kit (JDK) y otros paquetes vinculados que permiten su correcta ejecución.

Figura 75

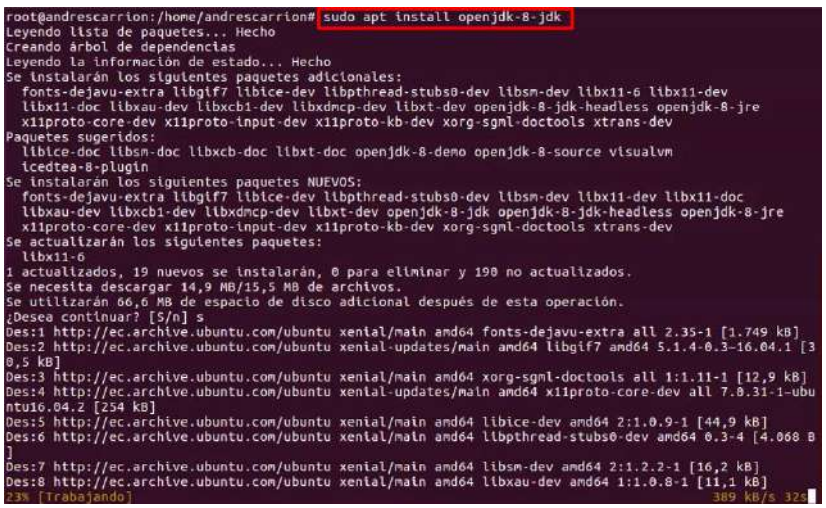
Instalación de componente ANT.



En la **Figura 76** se muestra el proceso de instalación del paquete openjdk-8-jdk, el cual es esencial para ejecutar aplicaciones desarrolladas en Java. Mediante el comando sudo apt install openjdk-8-jdk, se descargan e instalan tanto el Java Runtime Environment (JRE) como el Java Development Kit (JDK) correspondientes a la versión 8 de OpenJDK. Durante esta operación, también se incorporan paquetes adicionales necesarios para el entorno Java, incluidos componentes de desarrollo y bibliotecas gráficas.

Figura 76

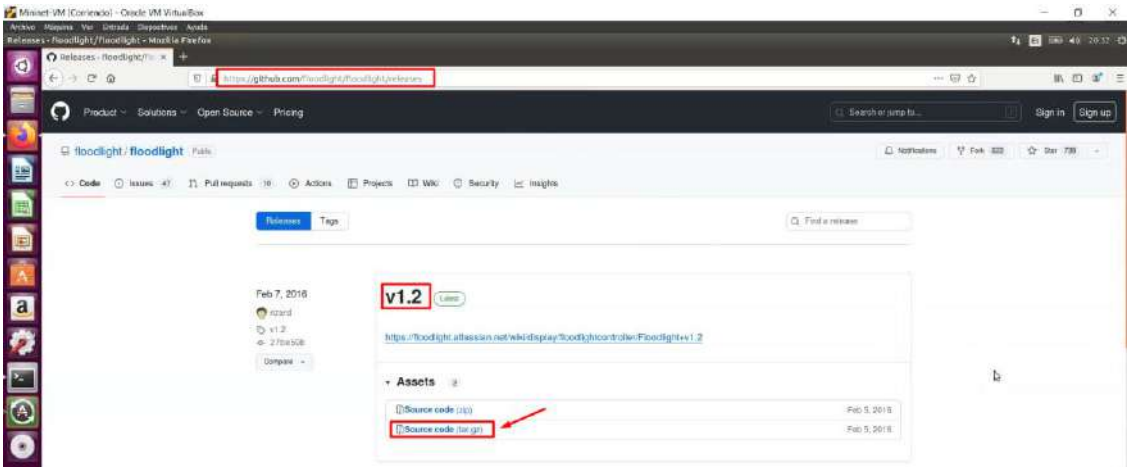
Instalación OpenJDK 8.



Luego, se descarga del controlador Floodlight, un controlador SDN (Software Defined Networking) de código abierto, desde la página oficial de GitHub, como se muestra en la **Figura 77.**

Figura 77

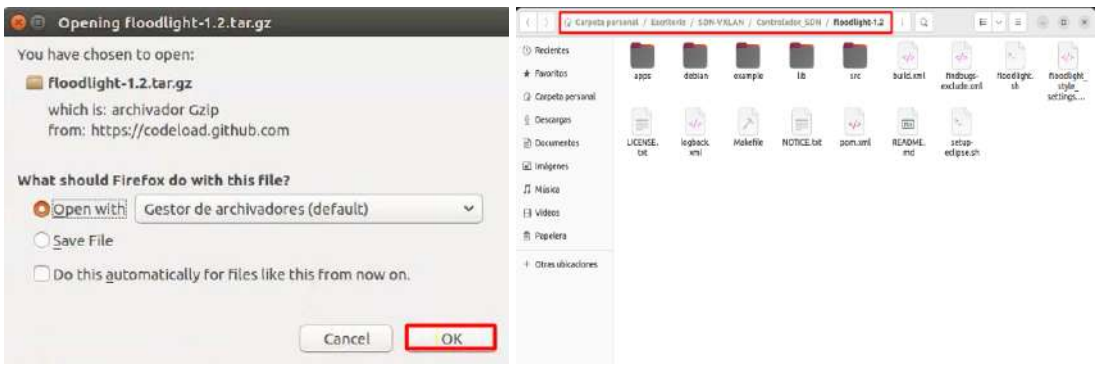
Descarga del archivo tar.gz del controlador Floodlight.



Se extrae el contenido del archivo `tar.gz` descargado y se la ubica en la zona del directorio que se crea conveniente para la posterior ejecución como en el ejemplo de la **Figura 78.**

Figura 78

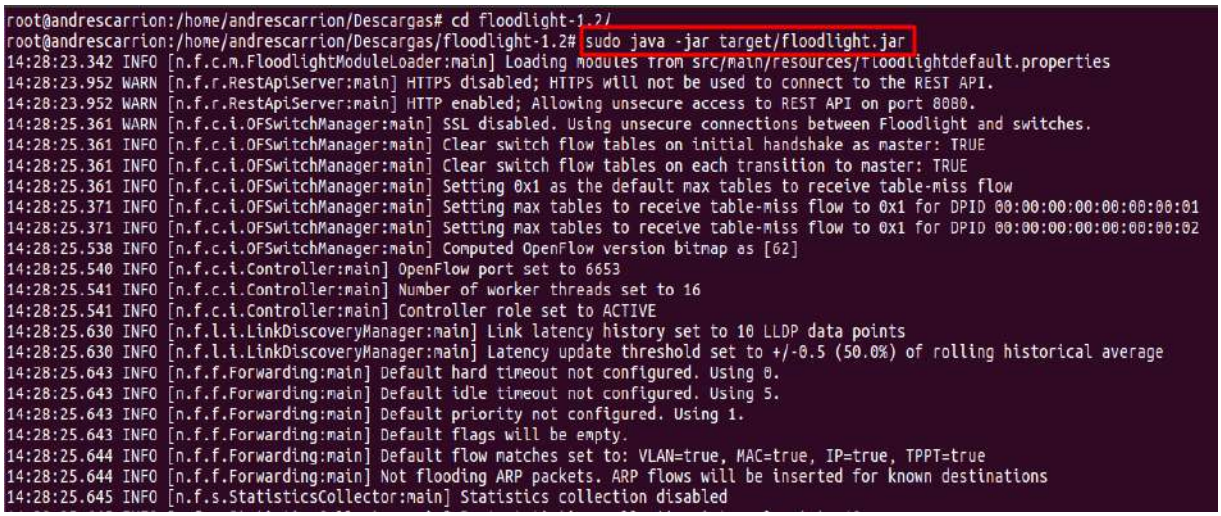
Descompresión y ubicación de la carpeta Floodlight-1.2.



Como último paso, como se muestra en la **Figura 79** se ingresa al controlador utilizando el comando sudo `java -jar target/floodlight.jar`, se ejecuta el archivo `JAR` de Floodlight con privilegios de superusuario, lo que permite acceder a las funcionalidades del controlador. Este procedimiento resulta clave para iniciar el funcionamiento del controlador, ya sea en entornos de desarrollo o producción, ya que permite gestionar la red de forma centralizada mediante la arquitectura SDN.

Figura 79

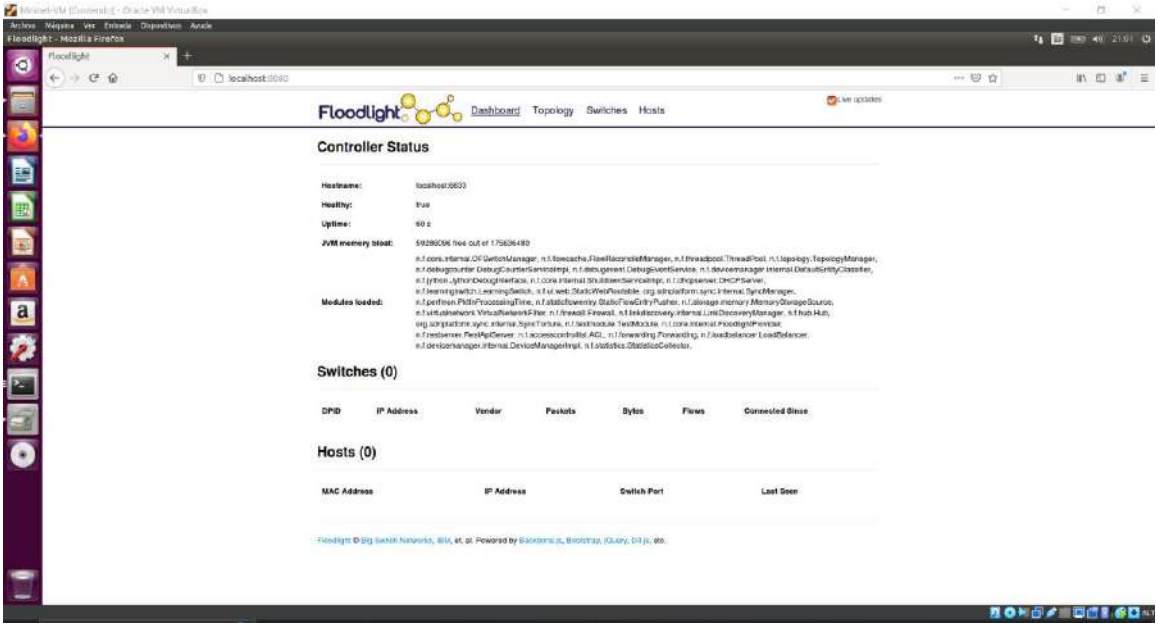
Iniciación del controlador SDN.



De igual manera, se puede observar la funcionalidad del controlador Floodlight mediante el acceso por el servidor HTTP con ayuda del navegador colocando la IP que maneja el controlador según se haya establecido seguido por el puerto correspondiente: `http://localhost:8080/ui/index.html` como en la **Figura 80.**

Figura 80

Interfaz del controlador desde el navegador.



En la **Tabla 17**, se presenta aspectos críticos del seleccionamiento del controlador Floodlight respecto al resto de controladores establecidos para el trabajo con redes SDN.

Tabla 17

Criterios clave para trabajar con Floodlight.

Criterio	Floodlight	Otros Controladores
Escalabilidad y Flexibilidad	Ofrece alta escalabilidad para redes grandes y complejas, flexible en la integración con VXLAN.	Algunos controladores pueden no manejar redes complejas con la misma eficiencia.
Soporte para OpenFlow	Soporte robusto y completo para OpenFlow, ideal para gestionar flujos en redes con VXLAN.	Ofrece un soporte robusto y completo para OpenFlow, lo que lo convierte en una opción ideal para la gestión de flujos de red en entornos que utilizan VXLAN.

Consumo de Recursos	Funciona de manera eficiente con bajos recursos, lo que lo hace adecuado para entornos con hardware limitado.	Puede requerir más recursos de hardware para manejar grandes redes y configuraciones avanzadas.
API RESTful	API RESTful sencilla y potente, lo que facilita la integración y automatización de redes con VXLAN.	La API RESTful puede no estar disponible o no ser tan accesible en otros controladores.
Soporte y Comunidad	Gran cantidad de documentación y comunidad activa, facilitando la resolución de problemas.	Otros controladores pueden tener menor soporte comunitario y menos documentación.
Modularidad y Extensibilidad	Modular y fácilmente extensible, permitiendo personalizaciones para VXLAN y otras configuraciones avanzadas.	Menos flexibilidad para añadir o modificar funcionalidades específicas.
Compatibilidad con Hardware	Amplia compatibilidad con switches físicos y virtuales que soportan VXLAN.	Algunos controladores pueden no tener compatibilidad tan amplia con hardware diverso.
Interoperabilidad con Redes Híbridas	Facilita la gestión de redes híbridas (físicas y virtuales) con VXLAN, permitiendo control centralizado.	Limitada capacidad para manejar redes híbridas de manera eficiente.
Soporte para VXLAN	Optimizado para configuraciones de VXLAN, permitiendo la extensión de redes L2 sobre L3 sin problemas.	Puede no soportar VXLAN de manera eficiente o con las mismas facilidades de integración.

Nota: La tabla destaca las razones por las que Floodlight es una buena elección para proyectos relacionados con redes SDN y VXLAN y compara a Floodlight con otros controladores SDN (OpenDaylight, ONOS, Ryu, Cisco ACI y VMware NSX).

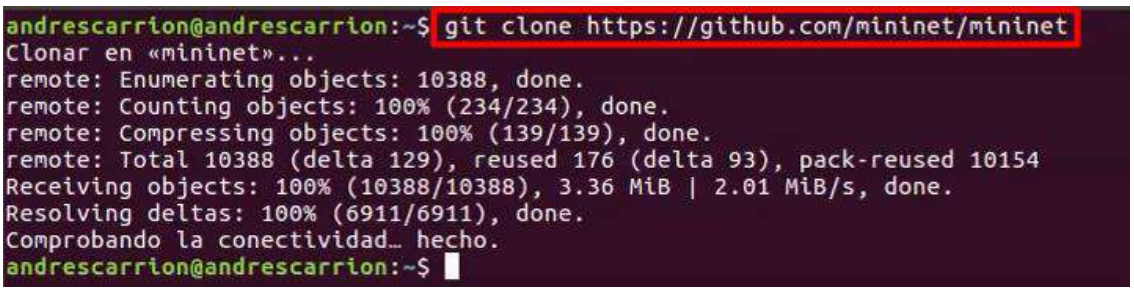
Instalación de MININET

En esta etapa de la instalación de Mininet, se utiliza el comando `git clone https://github.com/mininet/mininet` para clonar el repositorio oficial del software desde GitHub como en la **Figura 81.** Este proceso copia todos los archivos del proyecto al sistema local, incluidos los scripts de instalación, documentación, y demás componentes necesarios. Git comienza enumerando y comprimiendo los objetos del repositorio para optimizar la transferencia de datos, lo que asegura que se descarguen solo los elementos esenciales de forma eficiente.

Una vez que se reciben los objetos y se resuelven las diferencias para evitar duplicados, la clonación concluye con éxito mediante la verificación de la conectividad. Este proceso asegura que el usuario cuente con una versión completa y actualizada del software, lista para continuar con las siguientes etapas de la instalación. Además, este método facilita futuras actualizaciones del sistema, gracias a las capacidades de sincronización que ofrece Git.

Figura 81

Clonación del repositorio oficial de MININET.

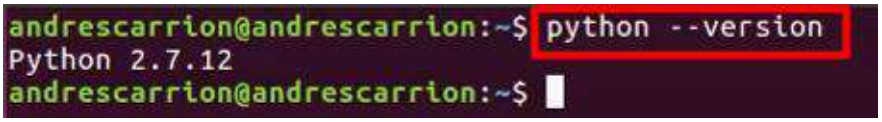


Python juega un papel esencial durante la instalación de Mininet, dado que muchos de sus componentes y scripts están desarrollados en este lenguaje. Aunque Python 2 ya no cuenta con soporte oficial, algunas versiones anteriores de Mininet todavía dependen de esta versión. Por esta razón, es importante verificar cuál versión de Python está instalada en el sistema para

asegurar que los pasos de instalación se ejecuten correctamente, tal como se muestra en la **Figura 82.**

Figura 82

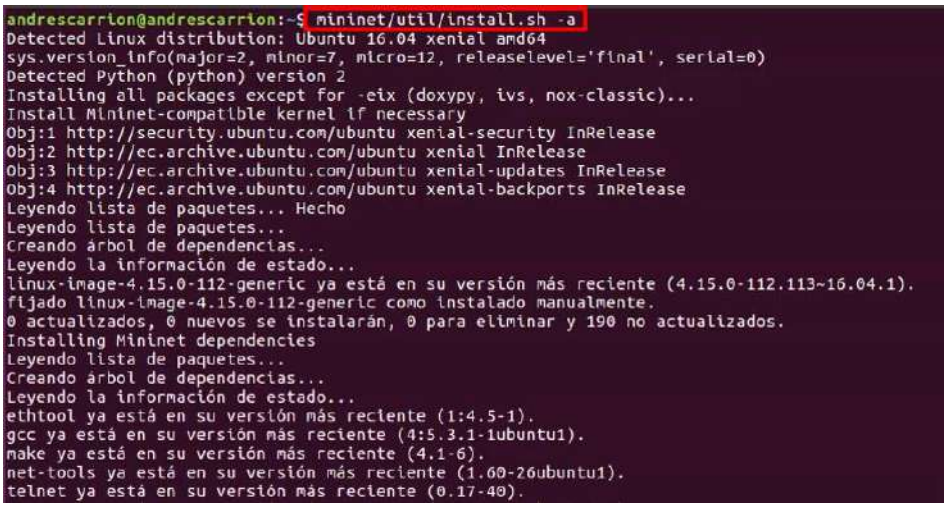
Verificación de la versión de Python disponible en el sistema operativo.



Finalmente, se da inicio al script de instalación de Mininet ubicado en la carpeta util del repositorio clonado que realizará una instalación completa de todos los paquetes y dependencias necesarias para garantizar que Mininet funcione correctamente como se muestra en la **Figura 83.** Este paso es crucial para asegurar que el entorno tenga todos los componentes necesarios para ejecutar Mininet sin inconvenientes.

Figura 83

Ejecución de script para instalación de MININET y sus componentes.



Finalmente, la **Figura 84** muestra la verificación de la herramienta Mininet. En el directorio `mininet/examples` se pueden encontrar diversos scripts que sirven como ejemplos

para realizar pruebas con diferentes topologías de red. Además, se destaca un editor gráfico que facilita la creación visual de estas topologías, permitiendo un diseño más intuitivo y práctico.

Figura 84

Interfaz gráfica de MININET.

Código Topología VXLAN – SDN

```
# -*- coding: utf-8 -*-
from mininet.net import Mininet  # Crear red Mininet
from mininet.node import RemoteController, OVSSwitch  # Controlador remoto y
switches OVS
from mininet.cli import CLI  # CLI interactiva
from mininet.link import TCLink  # Enlaces configurables
from mininet.log import setLogLevel  # Nivel de log

def create_network():
    # Instanciar red con controlador remoto y enlaces configurables
    net = Mininet(controller=RemoteController, switch=OVSSwitch, link=TCLink)

    # Agregar controlador remoto Floodlight
    controller = net.addController('c0', ip='127.0.0.1', port=6653)

    # Crear switches VTEP (Leaf switches)
    vtep1 = net.addSwitch('s3', cls=OVSSwitch)  # VNI 100
    vtep2 = net.addSwitch('s4', cls=OVSSwitch)  # VNI 200
    vtep3 = net.addSwitch('s5', cls=OVSSwitch)  # VNI 300

    # Crear hosts con IPs dentro del mismo rango
    hosts_vtep1 = [net.addHost(h, ip='10.0.0.{}'.format(i)) for i, h in
enumerate(['h1', 'h2', 'h3', 'h4'], start=1)]
    hosts_vtep2 = [net.addHost(h, ip='10.0.0.{}'.format(i)) for i, h in
enumerate(['h5', 'h6', 'h7', 'h8'], start=5)]
    hosts_vtep3 = [net.addHost(h, ip='10.0.0.{}'.format(i)) for i, h in
enumerate(['h9', 'h10', 'h11', 'h12'], start=9)]

    # Enlazar hosts a sus respectivos switches
    for h in hosts_vtep1:
        net.addLink(h, vtep1)
    for h in hosts_vtep2:
        net.addLink(h, vtep2)
    for h in hosts_vtep3:
        net.addLink(h, vtep3)

    # Iniciar red
    net.start()

    # Asignar IP loopback a cada VTEP (para VXLAN)
    vtep_ips = {'s3': '192.168.0.2', 's4': '192.168.10.2', 's5': '192.168.100.2'}
    for vtep, ip in vtep_ips.iteritems():
        net.get(vtep).cmd('ip addr add {}/32 dev lo'.format(ip))  # IP en loopback
        net.get(vtep).cmd('ip link set lo up')  # Activar loopback

    # Configuración de interfaces VXLAN entre VTEPs
    vxlan_config = [
        ('s3', 'vxlan301', '192.168.0.2', '192.168.10.2', 100),
        ('s3', 'vxlan302', '192.168.0.2', '192.168.100.2', 300),
        ('s4', 'vxlan401', '192.168.10.2', '192.168.0.2', 100),
        ('s4', 'vxlan402', '192.168.10.2', '192.168.100.2', 200),
        ('s5', 'vxlan501', '192.168.100.2', '192.168.10.2', 200),
        ('s5', 'vxlan502', '192.168.100.2', '192.168.0.2', 300)
    ]
```

```
    # Crear interfaces VXLAN en OVS
    for vtep, vxlan_iface, local_ip, remote_ip, vni in vxlan_config:
        net.get(vtep).cmd(
            'ovs-vsctl add-port {} {} -- set interface {} type=vxlan '
            'options:local_ip={} options:remote_ip={} options:key={}
options:dst_port=4789'.format(
                vtep, vxlan_iface, vxlan_iface, local_ip, remote_ip, vni))

    # Reglas OpenFlow para manejar tráfico VXLAN por VNI
    for vtep, vni in [(vtep1, 100), (vtep2, 200), (vtep3, 300)]:
        # Encapsular tráfico VLAN -> VNI
        vtep.cmd('ovs-ofctl add-flow {}
"priority=300,dl_vlan={},actions=set_field:{}->tun_id,output:NORMAL"'.format(vtep,
vni, vni))
        # Aceptar paquetes VXLAN entrantes
        vtep.cmd('ovs-ofctl add-flow {}
"priority=320,udp,tp_dst=4789,actions=NORMAL"'.format(vtep))
        # Desencapsular tráfico VXLAN -> normal
        vtep.cmd('ovs-ofctl add-flow {}
"priority=250,tun_id={},actions=strip_vlan,output=NORMAL"'.format(vtep, vni))

    # Regla adicional para permitir tráfico general entre VNIs
    for vtep in [vtep1, vtep2, vtep3]:
        vtep.cmd('ovs-ofctl add-flow {}
"priority=200,actions=NORMAL"'.format(vtep))

    # Mostrar flujos configurados en cada switch
    for vtep in [vtep1, vtep2, vtep3]:
        print "Flows en {}:".format(vtep)
        print vtep.cmd('ovs-ofctl dump-flows {}'.format(vtep))

    # Ejecutar pingall para aprendizaje de direcciones
    print("\n*** Ejecutando pingall para que el controlador aprenda la
topología...")
    net.pingAll()

    # Iniciar CLI interactiva
    CLI(net)

    # Detener la red al salir
    net.stop()

# Ejecutar con nivel de log 'info'
if __name__ == '__main__':
    setLogLevel('info')
    create_network()
```

Código Topología VXLAN – SDN – SPINE LEAF

```
# -*- coding: utf-8 -*-
from mininet.net import Mininet  # Crear red Mininet
from mininet.node import RemoteController, OVSSwitch  # Controlador remoto y OVS
from mininet.cli import CLI  # CLI de Mininet
from mininet.link import TCLink  # Enlaces configurables
from mininet.log import setLogLevel  # Nivel de log para Mininet

def create_network():
    # Crear la red Mininet con controlador remoto y switches OVS
    net = Mininet(controller=RemoteController, switch=OVSSwitch, link=TCLink)

    # Agregar el controlador remoto Floodlight
    controller = net.addController('c0', ip='127.0.0.1', port=6653)

    # Agregar switches Spine
    s1 = net.addSwitch('s1', cls=OVSSwitch)
    s2 = net.addSwitch('s2', cls=OVSSwitch)

    # Agregar switches Leaf (VTEPs)
    vtep1 = net.addSwitch('s3', cls=OVSSwitch)
    vtep2 = net.addSwitch('s4', cls=OVSSwitch)
    vtep3 = net.addSwitch('s5', cls=OVSSwitch)

    # Conectar Spine a Leaf
    for spine in [s1, s2]:
        for leaf in [vtep1, vtep2, vtep3]:
            net.addLink(spine, leaf)

    # Definir los hosts y asignarlos a VLANs
    vlan_10_hosts = ['h1', 'h2', 'h5', 'h6', 'h9', 'h10']
    vlan_20_hosts = ['h3', 'h4', 'h7', 'h8', 'h11', 'h12']

    # Crear hosts y asignarlos a cada VTEP
    hosts_vtep1 = [net.addHost(h) for h in ['h1', 'h2', 'h3', 'h4']]
    hosts_vtep2 = [net.addHost(h) for h in ['h5', 'h6', 'h7', 'h8']]
    hosts_vtep3 = [net.addHost(h) for h in ['h9', 'h10', 'h11', 'h12']]

    # Conectar los hosts a los switches Leaf
    for h in hosts_vtep1:
        net.addLink(h, vtep1)
    for h in hosts_vtep2:
        net.addLink(h, vtep2)
    for h in hosts_vtep3:
        net.addLink(h, vtep3)

    # Iniciar la red
    net.start()

    # Configurar interfaces Loopback en los VTEPs
    vtep_ips = {'s3': '192.168.0.2', 's4': '192.168.10.2', 's5': '192.168.100.2'}
    for vtep, ip in vtep_ips.items():
        net.get(vtep).cmd('ip addr add {}/32 dev lo'.format(ip))
        net.get(vtep).cmd('ip link set lo up')

    # Configurar interfaces VXLAN en los switches correctamente
```

```
    vxlan_config = [
        ('s3', 'vxlan301', '192.168.0.2', '192.168.10.2', 100),
        ('s3', 'vxlan302', '192.168.0.2', '192.168.100.2', 300),
        ('s4', 'vxlan401', '192.168.10.2', '192.168.0.2', 100),
        ('s4', 'vxlan402', '192.168.10.2', '192.168.100.2', 200),
        ('s5', 'vxlan501', '192.168.100.2', '192.168.10.2', 200),
        ('s5', 'vxlan502', '192.168.100.2', '192.168.0.2', 300)
    ]

    # Recorre cada configuración VXLAN definida en la lista vxlan_config
    for vtep, vxlan_iface, local_ip, remote_ip, vni in vxlan_config:
    # Ejecuta un comando en el nodo de Mininet identificado como 'vtep'.
    # El comando agrega un puerto de tipo VXLAN al switch OVS usando ovs-vsctl.
        net.get(vtep).cmd(
            'ovs-vsctl add-port {} {} -- set interface {} type=vxlan
options:local_ip={} options:remote_ip={} options:key={} options:dst_port=4789'
            .format(vtep, vxlan_iface, vxlan_iface, local_ip, remote_ip, vni))

          #  vtep,       ->  Nombre del switch OVS (VTEP)
          #  iface,      ->  Nombre de la nueva interfaz VXLAN
          #  iface,      ->  Se vuelve a usar para configurar la interfaz
          #  local_ip,   ->  Dirección IP del extremo local del túnel VXLAN
          #  remote_ip,  ->  Dirección IP del extremo remoto del túnel
          #  vni))       ->  VXLAN Network Identifier (VNI), clave del túnel

    # Configurar VLANs en los switches y los hosts
    for h in net.hosts:
    # Activa la interfaz principal eth0 del host (por defecto está down en Mininet)
        h.cmd('ip link set {}-eth0 up'.format(h.name))
    # Verifica si el host pertenece a la VLAN 10, sino pasa a VLAN 20
        if h.name in vlan_10_hosts:
            vlan_id = 10
        elif h.name in vlan_20_hosts:
            vlan_id = 20
        else:
            continue
        h.cmd('ip link add link {}-eth0 name {}-eth0.{} type vlan id
{}'.format(h.name, h.name, vlan_id, vlan_id)) # Crea una subinterfaz VLAN en el
host, ligada a eth0, con el ID correspondiente
        h.cmd('ip link set {}-eth0.{} up'.format(h.name, vlan_id))   # Activa la
subinterfaz VLAN recién creada
        h.cmd('ip addr add 10.0.{}.{} /24 dev {}-eth0.{}'.format(vlan_id, h.name[-
1], h.name, vlan_id))  # Asigna una dirección IP a la subinterfaz VLAN (ej.
10.0.10.1/24 o 10.0.20.2/24)

    # Agregar reglas OpenFlow para manejar VXLAN correctamente
    openflow_rules = [
        ('s3', 10, 100), ('s3', 20, 100), ('s3', 10, 300), ('s3', 20, 300),
        ('s5', 10, 200), ('s5', 20, 200)
    ]
    for switch, vlan, vni in openflow_rules: # Recorre todas las reglas y configura
los switches Open vSwitch (OVS)
        # Regla 1: Cuando llegue un paquete con cierta VLAN, se cambia a tun_id
(VNI) y se envía normalmente
        net.get(switch).cmd('ovs-ofctl add-flow {}
"priority=300,dl_vlan={},actions=set_field:{}-
>tun_id,output:NORMAL"'.format(switch, vlan, vni))
        # Regla 2: Acepta paquetes UDP destinados al puerto VXLAN (4789), se
procesa
```

```
        net.get(switch).cmd('ovs-ofctl add-flow {}
"priority=320,udp,tp_dst=4789,actions=NORMAL"'.format(switch))
        # Regla 3: Cuando llega tráfico encapsulado con tun_id (VNI), elimina VLAN
y lo reenvía
        net.get(switch).cmd('ovs-ofctl add-flow {}
"priority=250,tun_id={},actions=strip_vlan,output:NORMAL"'.format(switch, vni))

    # Realizar un pingall inicial para aprendizaje de direcciones en el controlador
    print("\n*** Ejecutando pingall para que el controlador aprenda la
topología...")
    net.pingAll()
    CLI(net)  # Iniciar CLI de Mininet
    net.stop()  # Detener la red al salir

if __name__ == '__main__':
    setLogLevel('info')
    create_network()
```

Código para escenario de pruebas (VxLAN – SDN – SPINE LEAF)

```
# -*- coding: utf-8 -*-
from mininet.net import Mininet  # Crear red Mininet
from mininet.node import RemoteController, OVSSwitch  # Controlador remoto y OVS
from mininet.cli import CLI  # CLI de Mininet
from mininet.link import TCLink  # Enlaces configurables
from mininet.log import setLogLevel  # Nivel de log para Mininet
import time  # Controlar tiempos de espera
import os  # Ejecutar comandos del sistema
import subprocess # Ejecutar comandos externos y manejar procesos
from subprocess import Popen, PIPE  # Ejecutar procesos en segundo plano


def create_network():
    # Crear la red Mininet con controlador remoto y switches OVS
    net = Mininet(controller=RemoteController, switch=OVSSwitch, link=TCLink)

    # Agregar el controlador remoto Floodlight
    controller = net.addController('c0', ip='127.0.0.1', port=6653)

    # Agregar switches Spine
    s1 = net.addSwitch('s1', cls=OVSSwitch)
    s2 = net.addSwitch('s2', cls=OVSSwitch)

    # Agregar switches Leaf (VTEPs)
    vtep1 = net.addSwitch('s3', cls=OVSSwitch)
    vtep2 = net.addSwitch('s4', cls=OVSSwitch)
    vtep3 = net.addSwitch('s5', cls=OVSSwitch)

    # Conectar Spine a Leaf
    for spine in [s1, s2]:
        for leaf in [vtep1, vtep2, vtep3]:
            net.addLink(spine, leaf)

    # Definir los hosts y asignarlos a VLANs
    vlan_10_hosts = ['h1', 'h2', 'h5', 'h6', 'h9', 'h10']
    vlan_20_hosts = ['h3', 'h4', 'h7', 'h8', 'h11', 'h12']

    # Crear hosts y asignarlos a cada VTEP
    hosts_vtep1 = [net.addHost(h) for h in ['h1', 'h2', 'h3', 'h4']]
    hosts_vtep2 = [net.addHost(h) for h in ['h5', 'h6', 'h7', 'h8']]
    hosts_vtep3 = [net.addHost(h) for h in ['h9', 'h10', 'h11', 'h12']]

    # Conectar los hosts a los switches Leaf
    for h in hosts_vtep1:
        net.addLink(h, vtep1)
    for h in hosts_vtep2:
        net.addLink(h, vtep2)
    for h in hosts_vtep3:
        net.addLink(h, vtep3)

    # Iniciar la red
    net.start()

    # Configurar interfaces Loopback en los VTEPs
    vtep_ips = {'s3': '192.168.0.2', 's4': '192.168.10.2', 's5': '192.168.100.2'}
    for vtep, ip in vtep_ips.items():
        net.get(vtep).cmd('ip addr add {}/32 dev lo'.format(ip))
```

```
        net.get(vtep).cmd('ip link set lo up')

    # Configurar interfaces VXLAN en los switches correctamente
    vxlan_config = [
        ('s3', 'vxlan301', '192.168.0.2', '192.168.10.2', 100),
        ('s3', 'vxlan302', '192.168.0.2', '192.168.100.2', 300),
        ('s4', 'vxlan401', '192.168.10.2', '192.168.0.2', 100),
        ('s4', 'vxlan402', '192.168.10.2', '192.168.100.2', 200),
        ('s5', 'vxlan501', '192.168.100.2', '192.168.10.2', 200),
        ('s5', 'vxlan502', '192.168.100.2', '192.168.0.2', 300)
    ]

    # Recorre cada configuración VXLAN definida en la lista vxlan_config
    for vtep, vxlan_iface, local_ip, remote_ip, vni in vxlan_config:
    # Ejecuta un comando en el nodo de Mininet identificado como 'vtep'.
    # El comando agrega un puerto de tipo VXLAN al switch OVS usando ovs-vsctl.
        net.get(vtep).cmd(
            'ovs-vsctl add-port {} {} -- set interface {} type=vxlan
options:local_ip={} options:remote_ip={} options:key={} options:dst_port=4789'
            .format(vtep, vxlan_iface, vxlan_iface, local_ip, remote_ip, vni))

          #  vtep,       ->  Nombre del switch OVS (VTEP)
          #  iface,      ->  Nombre de la nueva interfaz VXLAN
          #  iface,      ->  Se vuelve a usar para configurar la interfaz
          #  local_ip,   ->  Dirección IP del extremo local del túnel VXLAN
          #  remote_ip,  ->  Dirección IP del extremo remoto del túnel
          #  vni))       ->  VXLAN Network Identifier (VNI), clave del túnel

    # Configurar VLANs en los switches y los hosts
    for h in net.hosts:
    # Activa la interfaz principal eth0 del host (por defecto está down en Mininet)
        h.cmd('ip link set {}-eth0 up'.format(h.name))
    # Verifica si el host pertenece a la VLAN 10, sino pasa a VLAN 20
        if h.name in vlan_10_hosts:
            vlan_id = 10
        elif h.name in vlan_20_hosts:
            vlan_id = 20
        else:
            continue
        h.cmd('ip link add link {}-eth0 name {}-eth0.{} type vlan id
{}'.format(h.name, h.name, vlan_id, vlan_id)) # Crea una subinterfaz VLAN en el
host, ligada a eth0, con el ID correspondiente
        h.cmd('ip link set {}-eth0.{} up'.format(h.name, vlan_id))   # Activa la
subinterfaz VLAN recién creada
        h.cmd('ip addr add 10.0.{}.{} /24 dev {}-eth0.{}'.format(vlan_id, h.name[-
1], h.name, vlan_id))  # Asigna una dirección IP a la subinterfaz VLAN (ej.
10.0.10.1/24 o 10.0.20.2/24)

    # Agregar reglas OpenFlow para manejar VXLAN correctamente
    openflow_rules = [
        ('s3', 10, 100), ('s3', 20, 100), ('s3', 10, 300), ('s3', 20, 300),
        ('s5', 10, 200), ('s5', 20, 200)
    ]
    for switch, vlan, vni in openflow_rules: # Recorre todas las reglas y configura
los switches Open vSwitch (OVS)
        # Regla 1: Cuando llegue un paquete con cierta VLAN, se cambia a tun_id
(VNI) y se envía normalmente
```

```
        net.get(switch).cmd('ovs-ofctl add-flow {}
"priority=300,dl_vlan={},actions=set_field:{}-
>tun_id,output:NORMAL"'.format(switch, vlan, vni))
        # Regla 2: Acepta paquetes UDP destinados al puerto VXLAN (4789), se
procesa
        net.get(switch).cmd('ovs-ofctl add-flow {}
"priority=320,udp,tp_dst=4789,actions=NORMAL"'.format(switch))
        # Regla 3: Cuando llega tráfico encapsulado con tun_id (VNI), elimina VLAN
y lo reenvía
        net.get(switch).cmd('ovs-ofctl add-flow {}
"priority=250,tun_id={},actions=strip_vlan,output:NORMAL"'.format(switch, vni))

    # Realizar un pingall inicial para aprendizaje de direcciones en el controlador
    print("\n*** Ejecutando pingall para que el controlador aprenda la
topología...")
    net.pingAll()

# ---------------------- INICIAR PRUEBAS AUTOMÁTICAMENTE ---------------------- #

   # ---------------------- INICIAR SERVIDOR TCP EN H1 ---------------------- #
    h1 = net.get('h1')
    print("\nIniciando servidor TCP en h1...")
    h1.cmd('iperf -s -p 5001 &')

    time.sleep(5)   # Espera 5 segundos para asegurar que iperf -s ya está
escuchando

    #-------------------------------PRUEBA 1-------------------------------------#

    print("\n*** PRUEBA 1: Simulación de tráfico TCP con eventos...")

    # Iniciar captura en segundo plano
    pcap_path = "/home/andrescarrion/SDN_VXLAN/Pruebas/captura_prueba1.pcap" # Ruta
donde se guardará el archivo .pcap con el tráfico capturado
    # Ejecuta tcpdump en segundo plano para capturar paquetes TCP destinados al
puerto 5001
    os.system("sudo tcpdump -i any tcp and dst port 5001 -w
/home/andrescarrion/SDN_VXLAN/Pruebas/captura_prueba1.pcap &")
    print("\n*** Captura de tráfico iniciada con tcpdump...")

    # Asegurar que h1 tenga el servidor iperf corriendo
    net.get('h1').cmd('iperf -s -p 5001 &')

    # Lista de eventos con tiempos específicos
    eventos = [
      ('h4', 0, 'start'),   # h4 inicia tráfico en 0s
      ('h6', 10, 'start'),  # h6 inicia tráfico en 10s
      ('h8', 20, 'start'),  # h8 inicia tráfico en 20s
      ('h4', 30, 'stop'),   # h4 detiene tráfico en 30s
      ('h10', 30, 'start'), # h10 inicia tráfico en 30s
      ('h4', 40, 'start'),  # h4 reanuda tráfico en 40s
      ('h12', 40, 'start')  # h12 inicia tráfico en 40s
    ]

    start_time = time.time()  # Marca el tiempo de inicio de la prueba

    for host, event_time, action in eventos:
      time.sleep(max(0, event_time - (time.time() - start_time)))  # Calcula cuánto
tiempo esperar para que el evento suceda en el momento exacto
```

```
      if action == 'start':
        print("{} comienza a enviar paquetes TCP a h1...".format(host))
        net.get(host).cmd('iperf -c {} -p 5001 -t 50 &'.format(h1.IP()))  # Si la
acción es iniciar tráfico, se lanza iperf como cliente hacia h1
      elif action == 'stop':
        print("{} detiene el envío de paquetes TCP a h1...".format(host))
        net.get(host).cmd('pkill -f "iperf -c"')  # Si la acción es detener, se
finaliza cualquier cliente iperf corriendo en ese host

    # Esperar hasta los 50 segundos antes de detener tcpdump
    time.sleep(10)
    # Detiene tcpdump de manera segura enviando SIGINT (así se guarda bien el
archivo .pcap)
    os.system("sudo pkill -SIGINT tcpdump")

    # Asegurar que se detienen todos los procesos iperf
    os.system("sudo pkill -f 'iperf -c'")  # Mata todos los clientes iperf

    #-----------------------------------------------------------------

    print("\n*** Extrayendo datos con tshark...")

    # Definir ruta donde se guardarán los datos extraídos de la captura pcap
    datos_paquetes_path =
"/home/andrescarrion/SDN_VXLAN/Pruebas/datos_paquetes.txt"
    # Comando tshark para procesar el archivo .pcap
    comando_tshark = [
        "tshark",             # Ejecuta tshark
        "-r", pcap_path,           # Lee el archivo .pcap previamente capturado
        "-T", "fields",            # Salida en formato de campos simples
        "-e", "frame.time_epoch",  # Campo 1: timestamp (en formato epoch)
        "-e", "frame.len",         # Campo 2: tamaño del paquete (bytes)
        "-Y", "tcp.port == 5001"   # Filtro: solo tráfico TCP del puerto 5001
    ]

    # Ejecuta el comando y guarda la salida en un archivo de texto plano
    with open(datos_paquetes_path, "w") as f:
        subprocess.call(comando_tshark, stdout=f)

    print("Datos extraídos guardados en:", datos_paquetes_path)

    print("\n*** Procesando métricas desde datos_paquetes.txt...")

    archivo_metricas =
"/home/andrescarrion/SDN_VXLAN/Pruebas/metricas_por_segundo.txt" # Ruta donde se
guardarán las métricas por segundo (bytes y paquetes por segundo)

    datos_por_segundo = {} # Diccionario para agrupar los datos por segundo (clave
= segundo, valor = lista de tamaños de paquetes)

    # Leer línea por línea el archivo con datos de paquetes extraídos
    with open(datos_paquetes_path, "r") as f:
       for linea in f:
          linea = linea.strip()      # Elimina espacios al inicio y final
          if not linea:              # Si la línea está vacía, se ignora
            continue
          partes = linea.split()     # Divide en [timestamp, tamaño]
          if len(partes) != 2:       # Si no hay exactamente 2 columnas, ignora
```

```
            continue
          try:
            timestamp = float(partes[0])    # Convierte el timestamp (epoch)
            length = int(partes[1])         # Convierte el tamaño del paquete
(bytes)
          except:
            continue                        # Si hay error en la conversión, salta
línea
          segundo = int(timestamp)          # Redondea el timestamp a segundos
          if segundo not in datos_por_segundo:    # Si el segundo aún no está en el
diccionario, lo inicializa
            datos_por_segundo[segundo] = {"total_bytes": 0, "total_paquetes": 0}

          # Suma los bytes y los paquetes a ese segundo
          datos_por_segundo[segundo]["total_bytes"] += length
          datos_por_segundo[segundo]["total_paquetes"] += 1

    # Guardar métricas calculadas en un archivo de texto con columnas: segundo,
total_bytes, total_paquetes, avg_packet_size
    with open(archivo_metricas, "w") as f_out:
      f_out.write("segundo\ttotal_bytes\ttotal_paquetes\tavg_packet_size\n")  #
Escribe cabecera
      for segundo in sorted(datos_por_segundo.keys()):  # Recorre los segundos en
orden
        total_bytes = datos_por_segundo[segundo]["total_bytes"]
        total_paquetes = datos_por_segundo[segundo]["total_paquetes"]
        # Calcula el tamaño promedio del paquete para ese segundo
        if total_paquetes > 0:
            avg_packet_size = float(total_bytes) / total_paquetes
        else:
            avg_packet_size = 0

        # Escribe los valores en formato tabulado
        f_out.write("{}\t{}\t{}\t{:.2f}\n".format(segundo, total_bytes,
total_paquetes, avg_packet_size))

    print("Métricas calculadas y guardadas en:", archivo_metricas)

    print("\n*** PRUEBA 1 Finalizada.")


    #-----------------------------------PRUEBA 2-------------------------------#
    print("\n*** PRUEBA 2: Medición de RTT (ping de cada host a h1)...")
    # Definir la ruta del archivo de resultados generales
    rtt_general = "/home/andrescarrion/SDN_VXLAN/Pruebas/rtt_general.txt"

    # Borrar archivo previo para evitar acumulación de datos antiguos
    os.system("rm -f {}".format(rtt_general))

    h1 = net.get('h1')  # Definir el destino de los pings

    # Ejecutar ping desde cada host a h1 y guardar en un solo archivo
    for host in net.hosts:
        if host.name != 'h1':  # Excluir h1
            print("Midiendo RTT desde {} a h1...".format(host.name))

            # Ejecutar ping y guardar la salida en el archivo general
            host.cmd("ping -c 10 {} >> {} 2>&1".format(h1.IP(), rtt_general))
```

```
            # Agregar un separador entre los resultados de cada host
            with open(rtt_general, "a") as f:
                f.write("\n--- Fin de RTT desde {} ---\n\n".format(host.name))

    # Asegurar permisos de edición y eliminación
    os.chmod(rtt_general, 0o777)

    print("\n*** PRUEBA 2 Finalizada.")


    #-------------------------------PRUEBA 3-----------------------------------#
    print("\n*** PRUEBA 3: Medición de rendimiento (Throughput)...")

    # Archivo donde se almacenará todo el throughput en la carpeta de pruebas
    throughput_general =
"/home/andrescarrion/SDN_VXLAN/Pruebas/throughput_general.txt"

    # Borrar el archivo si ya existe para no acumular datos de pruebas anteriores
    os.system("rm -f {}".format(throughput_general))

    for host in net.hosts:
        if host.name != 'h1':  # Excluir h1 (servidor)
            print("{} está enviando tráfico a h1...".format(host.name))

            # Ejecutar iperf y guardar la salida en el archivo general sin archivos
individuales
            host.cmd("iperf -c {} -p 5001 -t 30 -i 1 >> {} 2>&1".format(h1.IP(),
throughput_general))

            # Agregar un separador entre los resultados de cada host para facilitar
la lectura
            with open(throughput_general, "a") as f:
                f.write("\n--- Fin de Throughput desde {} ---\n\n".format(host.name))

    # Asignar permisos para edición y eliminación
    os.chmod(throughput_general, 0o777)


    print("\n*** PRUEBA 3 Finalizada.")

    #-------------------------------------PRUEBA 4-------------------------------#
    print("\n*** PRUEBA 4: Análisis del impacto del balanceo de carga...")

    switches = ['s3', 's4', 's5', 's1', 's2']
    # Define una lista con los nombres de los switches donde se analizarán
estadísticas

    for host in net.hosts:
        if host.name != 'h1': # Para cada host distinto de h1,
            print("{} está enviando tráfico a h1...".format(host.name))  # Imprime
mensaje indicando que el host está enviando tráfico a h1
            host.cmd('iperf -c {} -p 5001 -t 30 &'.format(h1.IP()))   # Ejecuta en
el host un cliente iperf que envía tráfico TCP a h1 en el puerto 5001 durante 30
segundos en segundo plano

    time.sleep(10)  # Espera 10 segundos para dejar que el tráfico fluya y se
estabilice

    for switch in switches:
```

```
        print("\n*** Capturando estadísticas de balanceo de carga en
{}...".format(switch))  # Imprime mensaje indicando que va a capturar estadísticas
del switch actual
        stats_output = net.get(switch).cmd('ovs-ofctl dump-ports
{}'.format(switch)) # Ejecuta comando Open vSwitch para obtener estadísticas
detalladas de los puertos del switch
        with
open('/home/andrescarrion/SDN_VXLAN/Pruebas/balanceo_{}.txt'.format(switch), 'w')
as f:
            f.write(stats_output)  # Guarda la salida del comando en un archivo de
texto para análisis posterior

    print("\n*** PRUEBA 4 Finalizada.")

    #----------------------------------PRUEBA 5----------------------------------#
    print("\n*** PRUEBA 5: Simulación de congestión y análisis de Laberio...")

    # Saturar un enlace específico
    print("Generando congestión en el enlace de h2 a h1...")
    net.get('h2').cmd('iperf -c {} -p 5001 -t 120 -b 10M &'.format(h1.IP()))

    time.sleep(30)  # Esperar un tiempo para que la congestión se establezca

    # Capturar reglas de flujo antes del rebalanceo
    print("Capturando estado inicial de las reglas de flujo...")
    before_flows = net.get('s3').cmd('ovs-ofctl dump-flows s3')

    time.sleep(30)  # Permitir que Laberio actúe

    # Capturar reglas de flujo después del rebalanceo
    print("Capturando estado final de las reglas de flujo...")
    after_flows = net.get('s3').cmd('ovs-ofctl dump-flows s3')

    # Guardar resultados
    with open("/home/andrescarrion/SDN_VXLAN/Pruebas/laberio_before.txt", "w") as
f:
        f.write(before_flows)

    with open("/home/andrescarrion/SDN_VXLAN/Pruebas/laberio_after.txt", "w") as f:
        f.write(after_flows)

    print("\n*** PRUEBA 5 Finalizada.")

    # ---------------------- FIN DE PRUEBAS ---------------------- #

    CLI(net)  # Iniciar CLI de Mininet
    net.stop()  # Detener la red al salir

if __name__ == '__main__':
    setLogLevel('info')
    create_network()
```