UNIVERSIDAD ESTATAL
PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS
Y TELECOMUNICACIONES

CARRERA DE ELECTRÓNICA
Y TELECOMUNICACIONES

TRABAJO DE TITULACIÓN

Propuesta tecnológica, previo a la obtención del título de:

INGENIERO EN ELECTRÓNICA Y
TELECOMUNICACIONES

TEMA

"Desarrollo de escenarios de simulación en un entorno de virtualización de red para un servicio de video streaming multicast sobre MiniNet"

AUTOR

Quishpe Pacheco Edwin Daniel

PROFESOR TUTOR

Ing. Daniel Jaramillo Chamba, Mgt.

LA LIBERTAD – ECUADOR

2024

APROBACIÓN DEL TUTOR

En mi calidad de Tutor del trabajo de titulación denominado: "**Desarrollo de escenarios de simulación en un entorno de virtualización de red para un servicio de video streaming multicast sobre MiniNet**", presentado por **Quishpe Pacheco Edwin Daniel**, estudiante de la carrera de Electrónica y Telecomunicaciones de la Universidad Estatal Península de Santa Elena, me permito declarar que luego de haber orientado, estudiado y revisado, lo apruebo en todas sus partes y autorizo al estudiante para que inicie con los trámites legales correspondientes.

La libertad, 21 de octubre del 2024.

DANIEL ARMANDO JARAMILLO CHAMBA

Firmado digitalmente por DANIEL ARMANDO JARAMILLO CHAMBA

Ing. Daniel Jaramillo Chamba, Mgt.
DOCENTE TUTOR

AGRADECIMIENTO

Deseo expresar mi más sincero agradecimiento al Dios Altísimo por su infinita misericordia al concederme la vida, la salud y la sabiduría necesaria para llevar a cabo este trabajo de titulación, cuyo desarrollo ha representado un desafío constante y una fuente inagotable de aprendizaje.

Honro y dedico este trabajo a mis padres, José Quishpe y Elsy Pacheco, cuyo incansable esfuerzo, constancia y apoyo incondicional han sido pilares fundamentales durante todo este proceso.

Extiendo mi gratitud a la Universidad Estatal Península de Santa Elena y a la Facultad de Sistemas y Telecomunicaciones por su invaluable guía y apoyo constante durante mi formación académica.

MARANATHA...

Porque Jehová da la sabiduría, Y de su boca viene el conocimiento y la inteligencia.
Proverbios 2:6

Quishpe Pacheco Edwin Daniel

TRIBUNAL DE GRADO

Firmado electrónicamente por:
WASHINGTON DANIEL TORRES GUIN

Ing. Washington Torres Guin, Mgt.
DECANO DE LA FACULTAD

Firmado electrónicamente por:
JOSE MIGUEL SANCHEZ AQUINO

Ing. José Sánchez Aquino, Mgt.
DIRECTOR DE LA CARRERA

CARLOS EFRAIN ANDRADE CAICHO

Digitally signed by CARLOS EFRAIN ANDRADE CAICHO
DN: CN=CARLOS EFRAIN ANDRADE CAICHO, E=candrade5130@upse.edu.ec
Reason: Tesis Quishpe
Location: La Libertad
Date: 2024.11.05 09:16:15 -05'00'
Foxit PDF Reader Version: 2024.3.0

Ing. Carlos Andrade Caicho, Mgt.
DOCENTE DEL ÁREA

DANIEL ARMANDO JARAMILLO CHAMBA

Firmado digitalmente por DANIEL ARMANDO JARAMILLO CHAMBA

Ing. Daniel Jaramillo Chamba, Mgt.
DOCENTE TUTOR

Firmado electrónicamente por:
MARIA MARGARITA RIVERA GONZALEZ

Ab. María Rivera González, Mgt.
SECRETARIA GENERAL

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
CARRERA DE ELECTRÓNICA Y TELECOMUNICACIONES

"Desarrollo de escenarios de simulación en un entorno de virtualización de red para un servicio de video streaming multicast sobre MiniNet"

Autor: Quishpe Pacheco Edwin Daniel

Tutor: Ing. Daniel Jaramillo Chamba, Mgt.

RESUMEN

Este estudio se centra en la investigación y desarrollo de escenarios de simulación en un entorno de red virtualizado. Se implementan soluciones tecnológicas que integran la arquitectura de Redes Definidas por Software (SDN) mediante el Protocolo OpenFlow y un controlador centralizado. El objetivo es desplegar un servicio de video en streaming multicast en la plataforma de emulación de redes MiniNet. Para lograr esto, se emplearon los controladores de red OpenDayLight (ODL) y POX, así como el protocolo de gestión de grupos IGMPv3 para proporcionar capacidades de multidifusión. Mediante la virtualización de la red y la aplicación de los protocolos adecuados para la transmisión de datos, se exploran las oportunidades que las SDN pueden ofrecer para optimizar la transmisión de video en streaming y mejorar la administración de la red a través de una lógica centralizada y mediante programabilidad desde el plano de gestión. Los resultados obtenidos de los escenarios con diversas topologías demostraron cómo el aumento del retardo y la pérdida de paquetes pueden afectar significativamente la métrica de rendimiento de la red, produciendo retrasos y fluctuaciones en la calidad de servicio para el usuario final.

ABSTRACT

This study focuses on the research and development of simulation scenarios within a virtualized network environment. Technological solutions are implemented that incorporate the architecture of Software Defined Networks (SDN) through the OpenFlow Protocol and a centralized controller. The aim is to deploy a multicast streaming video service on the MiniNet network emulation platform. To achieve this, the OpenDayLight (ODL) and POX network controllers were used, as well as the IGMPv3 group management protocol to provide multicast capabilities. Through network virtualization and the application of the appropriate protocols for data transmission, the opportunities that SDNs can offer to optimize streaming video transmission and improve network management through centralized logic and programmability from the management plane are explored. The results obtained from scenarios with different topologies demonstrated how the increase in delay and packet loss can significantly affect the network performance metric, producing delays and fluctuations in the quality of service for the end user.

DECLARACIÓN DEL AUTOR

El contenido del presente Trabajo de Graduación es de mi responsabilidad; el patrimonio intelectual del mismo pertenece a la Universidad Estatal Península de Santa Elena.

Atentamente.

Quishpe Pacheco Edwin Daniel
AUTOR

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

En medio de la revolución digital, nos encontramos en un punto de inflexión donde la expansión de las redes de datos y el auge de los servicios de transmisión de video en streaming ha provocado un incremento considerable en el tráfico de datos, desafiando la capacidad de nuestras redes actuales y planteando problemas en términos de rendimiento, latencia y estabilidad que afectan la calidad de servicio y la gestión de la red.

En el contexto actual, las Redes Definidas por Software (SDN) se perfilan como una solución prometedora para la administración de servicios en tiempo real. A través de un controlador centralizado, las SDN proporcionan una visión integral del sistema, lo que permite una gestión de red automatizada y más eficaz. Por lo tanto, es imprescindible explorar las oportunidades que las SDN pueden ofrecer para optimizar la transmisión de video en streaming y mejorar la administración de la red. Esto a su vez, puede contribuir a proporcionar una mejor Calidad de Servicio (QoS) y Calidad de Experiencia (QoE) para el usuario final.

Este estudio se centra en la investigación y desarrollo de escenarios simulados en un entorno de virtualización de red. Se utilizan soluciones tecnológicas que implementan la arquitectura de Redes Definidas por Software (SDN) a través del Protocolo OpenFlow. El propósito es desplegar un servicio de video en streaming multicast en la plataforma de emulación de redes MiniNet. Para ello, se emplean los controladores de red OpenDayLight (ODL) y POX, así como el protocolo IGMPv3 para proporcionar capacidades de multidifusión en los escenarios de red desarrollados. De este modo, se podrán abordar los desafíos que surgen del crecimiento acelerado de los servicios de video en streaming, cumpliendo con los requisitos de red necesarios. Con esta investigación, se espera sentar las bases para futuras investigaciones en el campo de la virtualización de redes aplicadas a tecnologías de streaming.

CAPÍTULO I

1.1 ANTECEDENTES

La expansión de las redes de datos impulsada por el despliegue realizado a gran escala de servicios y tecnologías de transmisión de video ha ocasionado un vertiginoso crecimiento de la red de datos, permitiendo el amplio desarrollo de servicios de video en streaming y provocando un gran aumento de tráfico sobre las redes convencionales de hoy en día, debido a sus requisitos de ancho de banda se dificulta la gestión y optimización del rendimiento de la red. Además, para la transmisión de video es fundamental tener un buen rendimiento de la red para proporcionar calidad de servicio (QoS) y garantizar una calidad de la experiencia (QoE) adecuada para el usuario final. [1]

Resulta un gran reto, tanto para los operadores de red como para los proveedores de servicios de internet realizar la distribución de transmisiones multimedia en vivo dirigida a un gran número de usuarios finales a través de Internet. Un informe de Cisco VNI estimaba que el video IP representaría el 82 por ciento de todo el tráfico de Internet que los consumidores generaban a nivel mundial para 2022, frente al 75 por ciento en 2017. En general, se preveía que la transmisión de video en vivo crezca 15 veces entre 2017 y 2022 [2].

Se espera que las redes 5G emergentes habiliten un ecosistema de servicios que facilite nuevas oportunidades de negocio, apoyando también a los actores del mercado que no necesariamente posean una infraestructura de red, como proveedores de servicios y de aplicaciones. 5G ampliará aún más los volúmenes de tráfico debido a la adopción masiva de aplicaciones multimedia de contenido y servicios en la nube, introduciendo estrictos requisitos de servicio en áreas densas y en movimiento. Las redes 5G diversificarán los requisitos deseados en términos de rendimiento, latencia, estabilidad, etc. A medida que servicios multimedia de video streaming dominan el mercado, existe un número cada vez mayor de Proveedores de Servicio de Video (VSP) como Netflix, Amazon, YouTube, etc. Se espera que esto contribuya a aumentar el crecimiento del tráfico de video IP, no obstante, también fuera del negocio del entretenimiento existen varios servicios que dependen del video IP, como salud, electrónica, seguridad, protección, etc. [3]

Se han realizado trabajos de investigación donde se realizan estudios de simulación. En una investigación se desarrolló escenarios simulados para la distribución de contenido multimedia sobre el emulador MiniNet, mediante el empleo de topologías de redes SDN y especificando los parámetros necesarios para generación de tráfico multimedia, así como medición del rendimiento de la red. En consecuencia, permitió el análisis y estudio del comportamiento de la red en tiempo real, así como los efectos que se derivan del tráfico multimedia y la congestión de la red. [4]

1.2 DESCRIPCIÓN DEL PROYECTO

Se realizará el desarrollo de escenarios de simulación en un entorno de virtualización de red utilizando el protocolo OpenFlow para transmisión de video streaming sobre la plataforma MiniNet, el cual consiste en tres fases:

1.2.1 FASE DE INVESTIGACIÓN Y RECOLECCIÓN DE INFORMACIÓN:

Primero se realizará la investigación de los principales principios técnicos y fundamentos teóricos necesarios para poder comprender el funcionamiento de la tecnología de virtualización de red, así como sus beneficios respecto a tecnologías de red convencionales, permitiendo reconocer el potencial de utilizar la virtualización para admitir múltiples servicios como el video streaming, de igual manera se describirá los fundamentos principales de la tecnología streaming de video.

1.2.2 FASE DE DISEÑO Y DESARROLLO:

Se realizará el diseño de escenarios de simulación en un entorno de virtualización de red, mediante redes definidas por software utilizando el protocolo OpenFlow para la transmisión del video por protocolo de internet IP. Se trabajará sobre el software open source MiniNet el cual es ampliamente utilizado en la investigación, desarrollo y experimentación con sistemas de redes definidas por software. Los escenarios a desarrollar consisten en una topología de red que se basa en una arquitectura SDN y se compone de: un servidor de streaming, un controlador de red, un conmutador habilitado para OpenFlow y estaciones cliente. Bajo esta topología de red se desplegará un servicio de video streaming de tipo Multicast y se desarrollará en distintos escenarios para varias estaciones cliente. Además, se utilizará el controlador de red OpenDaylight y en cada escenario se

pondrá a prueba los protocolos de transmisión RTSP y RTMP permitiendo evaluar la virtualización de red en un servicio de video streaming.

1.2.3 FASE DE EVALUACIÓN DE RESULTADOS:

Mediante la ejecución de los escenarios desarrollados en MiniNet, se podrá realizar una evaluación de los resultados obtenidos de métricas y parámetros de red necesarios para un desempeño óptimo del servicio de video streaming. Las métricas de red a evaluar que se extraerán de cada escenario son: el ancho de banda (BW), rendimiento de la red, pérdida de paquetes, variación del retardo (Jitter). Además, se empleará la herramienta de captura de tráfico Wireshark con el fin de identificar los protocolos presentes en los escenarios desarrollados, esto a su vez permitirá extraer datos que sirvan para evaluar la tecnología de virtualización de red mediante un estudio de simulación.

1.3 OBJETIVOS DEL PROYECTO

1.3.1 OBJETIVO GENERAL

Desarrollar escenarios de simulación en un entorno de virtualización de red, utilizando el protocolo OpenFlow para un servicio de video streaming multicast sobre la plataforma MiniNet.

1.3.2 OBJETIVOS ESPECÍFICOS

- Describir los principales fundamentos de la virtualización de red y el streaming de video.
- Diseñar escenarios de simulación de red virtual en un entorno SDN/OpenFlow, para la transmisión de video streaming de tipo multicast sobre el software open source MiniNet.
- Evaluar los resultados obtenidos de las métricas y parámetros de red necesarios para un óptimo desempeño del servicio de video streaming.

1.4 RESULTADOS ESPERADOS

Se espera que al concluir con la investigación y el desarrollo del estudio de simulación se obtengan los siguientes resultados, los cuales se derivan de un estudio de simulación:

- Programabilidad de la red otorgando flexibilidad de la carga de trabajo mediante virtualización de red, para un óptimo desempeño en la prestación de un servicio de video streaming.

- Control centralizado de la red permitiendo gestión integral y administración consistente, aplicando una arquitectura de red definida por software (SDN) donde se despliega un servicio de video streaming.
- Escenarios de red desarrollados en un estudio de simulación sobre MiniNet, que permitan enfrentar a las necesidades y demandas cambiantes requeridas para el video streaming.
- Reducción de costos en el diseño de redes de hardware propietario mediante abstracción de la red, al tener una red basada software que cuenta con los beneficios que otorga la virtualización de red.

1.5 JUSTIFICACIÓN

La virtualización de red (NV) implica la transformación de una red que anteriormente requería de elementos de hardware físico en una red virtual basada en software. El primordial objetivo que recae en la virtualización de red, es proveer de una capa de abstracción entre aplicaciones de software, hardware físico y servicios que se ejecutan en esta tecnología abarcando los diferentes tipos de virtualización de las TIC. Puede utilizarse de tal forma que se pueda hacer más robustas varias redes físicas fortaleciendo el desempeño y gestión de la red. Los proveedores de servicios digitales pueden usar la virtualización de red para optimizar los recursos de hardware, de servidores e infraestructura de red, reducir costos en elementos físicos, permitir derivar funciones que antes requerían de costosos elementos de hardware y en particular optimizar de manera flexible la velocidad, tiempo de respuesta, confiabilidad y escalabilidad de la red. [5]

Las redes definidas por software (SDN) son una alternativa ideal para administrar servicios que operan en tiempo real, porque otorgan una perspectiva global a través de todo el sistema en un controlador centralizado permitiendo que se realice una gestión de la red de manera automatizada, empleando la información recibida de monitoreo de los conmutadores de manera más inteligente. Además, varios controladores pueden ser vinculados permitiendo incrementar la escalabilidad de la red, reducir el tiempo empleado en la resolución de problemas aprovechando la abstracción y programabilidad de la red. [6]

La transmisión de video constituye la mayor parte del tráfico de red en la Internet actual. Solo YouTube representa más del 20 % del tráfico descendente en las redes

móviles de América del Norte, lo que constituye la mayor fuente individual de tráfico descendente. Las redes de entrega de contenido (CDN) como Akamai o Google Global Cache ayudan a entregar contenido a nivel mundial. Al proporcionar cachés cerca de los usuarios, el proveedor de contenido se libera de la carga de tráfico y se puede lograr un tiempo de almacenamiento en búfer de video inicial bajo. Los CDN se utilizan por lo general, para la transmisión de video bajo demanda y en streaming. SDN para la gestión de redes utilizadas en la transmisión de video streaming es un área amplia de investigación, la cual posee mucho potencial que puede ayudar en nuevos mecanismos e incluso métricas para detectar y superar problemas de calidad de servicio (QoS) hacia los usuarios finales. [7]

Con la investigación y desarrollo de la presente solución tecnológica, se pretende que sirva como base para desarrollar nuevas investigaciones referentes a la virtualización de red aplicadas a tecnologías de streaming. Se hace especial énfasis en la utilización de redes definidas por software SDN para el despliegue de un servicio de video streaming. Permitiendo la aplicación y proliferación de estas tecnologías para atender a la creciente demanda con los requisitos de red necesarios. De esta forma enfrentar los problemas que surgen del rápido crecimiento de los servicios de video streaming, los cuales se presta a través de redes tradicionales donde las capacidades de red se encuentran limitadas.

1.6 ALCANCE

El objetivo de la presente investigación es evaluar la virtualización de red y el video streaming, teniendo en cuenta diversas métricas de rendimiento. Se espera desplegar un servicio de video streaming de tipo Multicast con IGMPv3 en múltiples escenarios de emulación de red, con diferentes topologías de red: lineal, árbol y malla de manera estable y libre de errores, utilizando la plataforma Mininet. Para lograr esto, se utilizará un entorno de Red Definida por Software (SDN) como arquitectura de red, el protocolo de comunicaciones OpenFlow que es un estándar abierto para la investigación en trabajos con SDN, los protocolos de transmisión de medios RTSP, RTMP, RTP, UDP y los controladores de red OpenDayLight y POX.

En cada escenario desarrollado, se evaluarán las métricas de rendimiento de la red, que incluyen: Latencia, throughput para TCP y UDP, la pérdida de paquetes y la variación del retardo (Jitter). Los escenarios de simulación serán evaluados para varias estaciones cliente, las cuales integrarán el reproductor multimedia VLC para mostrar el video streaming en tiempo real.

Además, se utilizará la herramienta Wireshark para capturar el tráfico y reconocer los protocolos que intervienen en los escenarios de red desarrollados. De esta manera, cada escenario proporcionará datos que permitirán evaluar principalmente la tecnología de virtualización de red donde opera un servicio de video streaming. Esto permitirá identificar factores clave para comprender los beneficios y desventajas que estas tecnologías ofrecen en la actualidad.

1.7 METODOLOGÍA

Para el trabajo de investigación es necesario que se fundamente las opciones metodológicas y el proceso que se siguió durante la ejecución del estudio de simulación. La metodología de investigación que más se ajusta a los objetivos planteados y a la resolución del problema es la investigación empírica, la cual se basa en evidencia verificable y concreta. Emplea métodos cuantitativos y cualitativos para compilar información. [8]

Se obtendrán las respectivas conclusiones como resultado de la experimentación y obtención de resultados de los escenarios de simulación desarrollados sobre el software MiniNet, permitiendo de esta forma la resolución del proyecto.

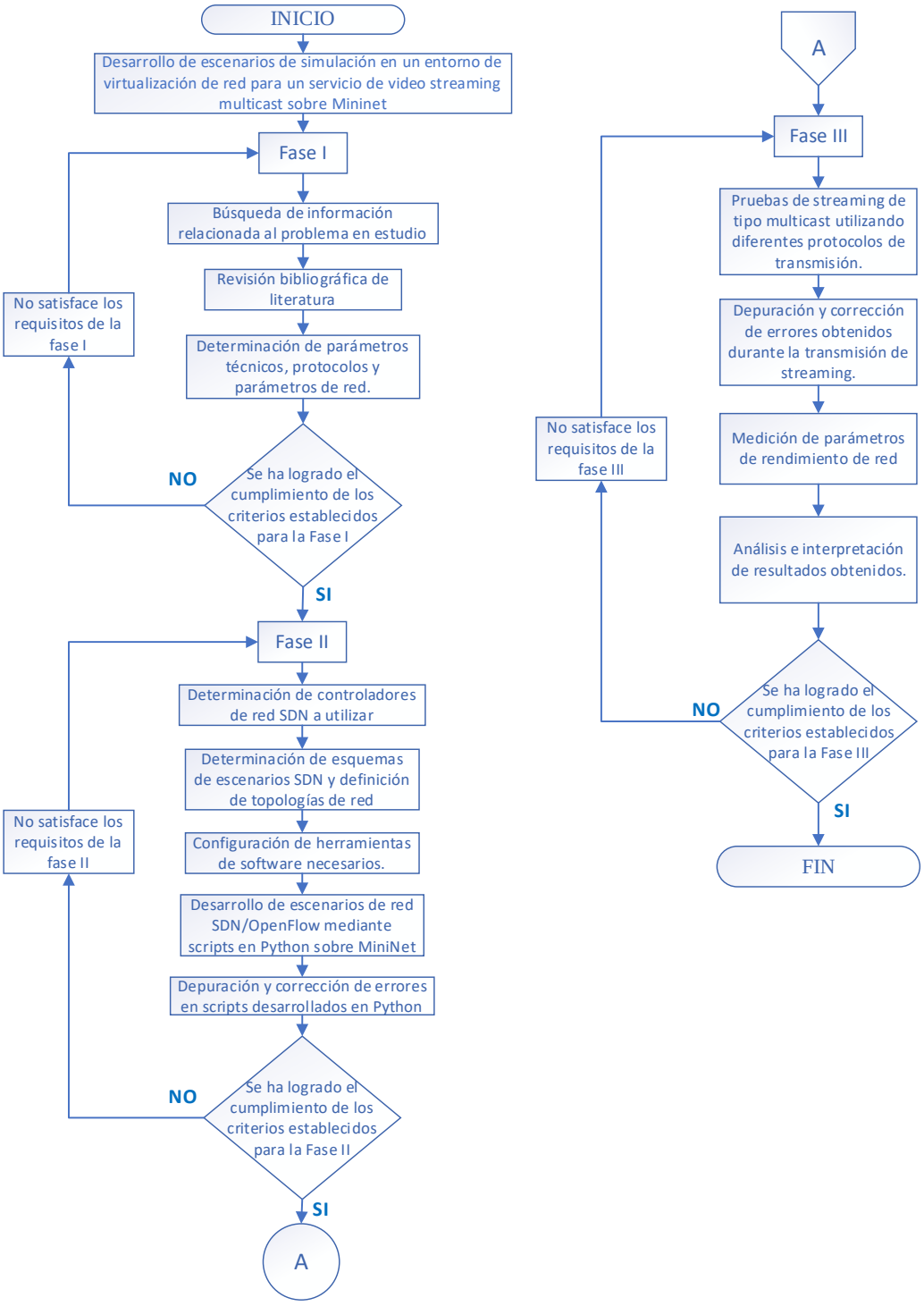


Figura 1. *Diagrama de bloques de la metodología propuesta.*
Fuente: *Elaborado por el Autor.*

CAPÍTULO II

2.1 MARCO CONTEXTUAL

La transmisión de video en tiempo real, conocido como streaming, se ha consolidado como un método predominante para la distribución de contenido multimedia. El streaming de video implica la transmisión ininterrumpida de datos de video a través de una red, lo que permite a los usuarios acceder al contenido sin la necesidad de descargarlo en su totalidad. Sin embargo, este proceso presenta desafíos significativos en términos de calidad de transmisión, sincronización de datos y gestión de recursos de red.

Una limitación notable en las redes tradicionales es la falta de programabilidad. Las configuraciones y políticas de red se implementan generalmente directamente en los dispositivos de red, lo que dificulta la automatización y gestión eficiente de la red. Esta falta de programabilidad restringe la capacidad de adaptación de la red a las necesidades cambiantes y obstaculiza la implementación de nuevas funcionalidades. Por lo tanto, es imperativo explorar soluciones que permitan una mayor flexibilidad y control sobre las operaciones de red. [9]

Las redes tradicionales también plantean un desafío considerable en términos de escalabilidad. Con el crecimiento exponencial de los dispositivos y servicios conectados a la red, la gestión y escalabilidad eficiente de las infraestructuras de red existentes se vuelve cada vez más compleja. Esta dificultad se debe en gran medida a la falta de una separación clara entre el plano de control y el plano de datos, lo que obstaculiza la distribución y escalado eficiente de los servicios. Por lo tanto, es crucial investigar enfoques que permitan una mayor flexibilidad y control sobre las operaciones de red para superar estos desafíos. [10]

El desarrollo de escenarios de simulación en un entorno de virtualización de red, utilizando el protocolo OpenFlow para un servicio de transmisión de video multicast en la plataforma MiniNet, se sitúa en el contexto de las Redes Definidas por Software (SDN). OpenFlow, que actúa como protocolo de comunicación entre el plano de control y el plano de datos en una arquitectura SDN, permite una gestión centralizada y programable de los dispositivos de red. Esta característica es particularmente útil en escenarios de transmisión de video multicast, donde se requiere la transmisión eficiente de contenido multimedia a múltiples receptores.

La plataforma MiniNet, un emulador de red de código abierto, proporciona un entorno flexible para la creación de topologías de red virtuales y la simulación de interacciones entre los diferentes elementos de la red. Este enfoque permite explorar y evaluar diversas estrategias y técnicas para mejorar la eficiencia y escalabilidad del servicio de transmisión de video multicast en un entorno SDN.

El desarrollo de escenarios de simulación permitirá la evaluación de diversas métricas de red fundamentales que influyen en el rendimiento del servicio. Estas métricas proporcionarán una visión detallada de la eficiencia y efectividad de las estrategias implementadas. Además, se utilizará la herramienta de captura de tráfico Wireshark para obtener una comprensión integral de los protocolos involucrados en los escenarios de simulación. Esta herramienta proporcionará información detallada sobre el comportamiento de la red, lo que permitirá un análisis más profundo y una optimización más efectiva del rendimiento del servicio.

Es importante destacar que este enfoque no sólo permitirá mejorar el rendimiento del servicio, sino que también contribuirá a la comprensión general de las redes definidas por software y su potencial para mejorar la eficiencia y escalabilidad de los servicios de red.

2.2 MARCO CONCEPTUAL

2.2.1 VIRTUALIZACIÓN:

La virtualización es un concepto fundamental en las tecnologías de la información y comunicación que permite combinar o dividir recursos informáticos, ya sean elementos físicos o de software, para crear uno o varios entornos operativos. Este proceso se realiza mediante diversas metodologías, como la partición de hardware y software, la simulación parcial o completa de máquinas, la emulación, el tiempo compartido, entre otros. Las tecnologías de virtualización tienen aplicaciones significativas en una amplia gama de áreas, incluyendo servidores, plataformas informáticas seguras, soporte para múltiples sistemas operativos, depuración y desarrollo del kernel, migración del sistema y automatización de la red. Aunque la mayoría de estas tecnologías presentan entornos operativos similares al usuario

final, tienden a variar considerablemente en los niveles de abstracción en los que operan y la arquitectura subyacente. [11]

Por lo tanto, es esencial comprender las características y capacidades específicas de cada tecnología de virtualización para poder seleccionar e implementar la más adecuada para cada aplicación específica. Esta comprensión permitirá a los profesionales de las tecnologías de la información y comunicación aprovechar al máximo las ventajas que ofrece la virtualización, mejorando así la eficiencia y flexibilidad de sus sistemas informáticos.

2.2.1.1 TIPOS DE VIRTUALIZACIÓN:

Dentro del ámbito de la virtualización, existen diversos tipos que se aplican a los sistemas informáticos. En el contexto de esta investigación, se abordarán algunos conceptos fundamentales. Estos conceptos proporcionarán una base sólida para entender la naturaleza y las aplicaciones de la virtualización en los sistemas informáticos modernos.

2.2.1.2 VIRTUALIZACIÓN DE RED (NV):

Las redes físicas están siendo virtualizadas para permitir la creación de particiones de red lógicas y proporcionar capacidades de infraestructura como servicio (IaaS) a través de interfaces de programación abiertas. Mediante la virtualización, los recursos de red pueden ser particionados o agregados en cualquier configuración deseada para formar redes virtuales, las cuales están desacopladas de sus redes físicas subyacentes. Esto permite que las aplicaciones y la infraestructura de red física puedan evolucionar de manera independiente, ofreciendo una flexibilidad operativa significativa tanto para los operadores de red como para sus clientes. [12]

La virtualización de red puede ser conceptualizada como el proceso de desacoplamiento de las funciones tradicionalmente asociadas a los proveedores de servicios de Internet (ISP). Este proceso se realiza dividiendo las responsabilidades en dos entidades distintas e independientes: los proveedores de infraestructura (InP) y los proveedores de servicios (SP). Los proveedores de infraestructura (InP) son responsables de la administración y mantenimiento de la infraestructura física. Por otro lado, los proveedores de servicios (SP) se encargan de la creación de redes virtuales (VN). Esto se logra mediante la agregación de

recursos provenientes de múltiples InP, permitiendo así la oferta de servicios integrales de extremo a extremo. [13]

Este ambiente favorece el despliegue y coexistencia de arquitecturas de red heterogéneas, liberándolas de las limitaciones inherentes a la estructura actual de Internet. De esta manera, se promueve una mayor flexibilidad y adaptabilidad en la prestación y consumo de servicios en línea.

2.2.1.3 VIRTUALIZACIÓN DE FUNCIONES DE RED (NFV):

La Virtualización de Funciones de Red (NFV, por sus siglas en inglés) representa un paradigma emergente en el ámbito de la industria de las telecomunicaciones para proporcionar agilidad y flexibilidad en el despliegue de servicios de red. Este enfoque innovador permite la transformación de las funciones de red tradicionalmente basadas en hardware, en funciones de red virtualizadas. La NFV se emplea para virtualizar las funciones de red, desacoplando la implementación del software del hardware dedicado. Este proceso permite una mayor flexibilidad y escalabilidad, al permitir que las funciones de red se implementen y operen independientemente del hardware subyacente. [14]

Además, la NFV está ganando un considerable impulso entre los operadores de red. Esto se debe a la posibilidad de implementar la versión de software de un dispositivo de hardware, conocida como función de red virtual (VNF), en productos básicos y servidores de propósito general. Esta capacidad para desplegar y gestionar funciones de red a través de software, en lugar de hardware dedicado, ofrece ventajas significativas en términos de costos, eficiencia y agilidad operativa.

La red consta de varias cajas intermedias de hardware. como cortafuegos, equilibrador de carga, sistema de detección de intrusos (IDS), Sistema de prevención de intrusiones (IPS), Traducción de dirección de red (NAT), para proporcionar diferentes servicios de red a los usuarios finales. Estos dispositivos de red de hardware (middleboxes) ofrecen una red con alto rendimiento y confiabilidad debido a su implementación mediante un protocolo distribuido. Sin embargo, la gestión e integración de estos servicios de red en el sistema actual conlleva altos costos debido a la arquitectura propietaria de estos dispositivos de hardware. [15]

La virtualización de funciones de red (NFV) es un paso innovador para eliminar estos problemas con la ayuda de la tecnología de virtualización. NFV hace que la red sea ágil, flexible y reduce el costo al separar la instancia de software de la de hardware dedicado subyacente y lo implementa en el servidor comercial listo para usar. Como consecuencia, no hay necesidad de comprar nuevos equipos para desplegar estos servicios.

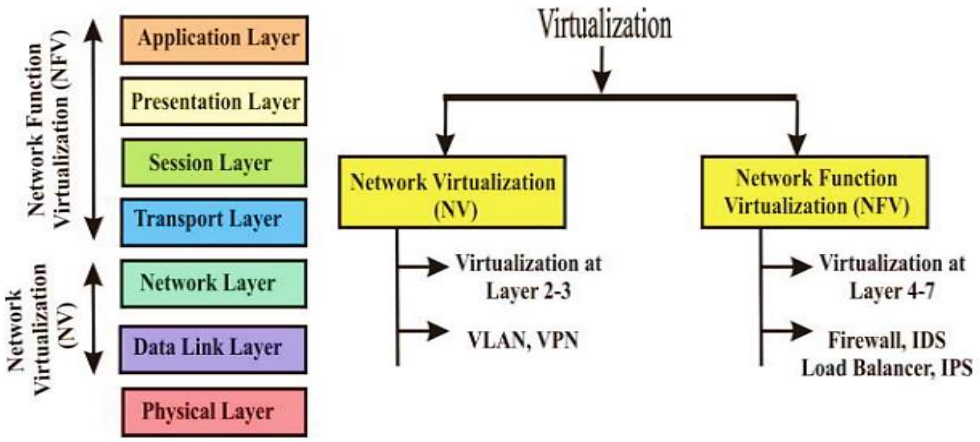


Figura 2. *Virtualización de Red (NV) Vs. Virtualización de Funciones de Red (NFV)*

Fuente: *Imagen extraída del sitio web IEEE Xplore, Architectural Framework, Research Issues and Challenges of Network Function Virtualization .*

2.2.2 REDES DEFINIDAS POR SOFTWARE (SDN):

El paradigma Redes Definidas por Software (SDN) se ha definido para convertir el entorno de red en uno nuevo, más inteligente y adaptable, para admitir nuevas aplicaciones. Este nuevo paradigma introduce un nodo centralizado, denominado Controlador de red definido por software (SDN-C), que elimina la integración vertical de las redes heredadas. Este enfoque aumenta la flexibilidad y simplifica la gestión de la red al desacoplar el plano de control y el plano de datos. Los nodos de red son programables por el SDN-C a través del protocolo OpenFlow. La programabilidad de los nodos de red se logra mediante la introducción de niveles adecuados de abstracción, a los cuales se puede acceder mediante el uso de interfaces de control o a través de una interfaz de programación de aplicaciones (API). [16]

Para resumir, la definición original de SDN es simple de decir: Una red en la que el plano de control está físicamente separado del plano de reenvío y un único plano de control controla varios dispositivos de reenvío. Otra forma de enmarcar SDN es pensar que tiene dos fases. En la Fase 1, los operadores de red tomaron posesión

del plano de control y ahora, en la Fase 2, toman el control de cómo se procesan los paquetes en el plano de datos. La Fase 2 todavía es un trabajo en progreso, pero como postula Nick McKeown, el estado final aspiracional es uno en el que: "Las redes [con suerte] serán programadas por muchos y operadas por pocos". Es decir, SDN no se trata solo de cambiar el control de los proveedores a los operadores, sino que en última instancia se trata de cambiar el control de los proveedores a los operadores y a los usuarios. Ese es el objetivo a largo plazo, inspirado en lo que los servidores comerciales y el software de código abierto hicieron por la industria informática. [17]

Las redes definidas por software (SDN) son un enfoque de cómo implementamos las redes, lo cual es importante porque afecta el ritmo de la innovación. SDN no aborda directamente ninguno de los desafíos técnicos de enrutamiento, control de congestión, ingeniería de tráfico, seguridad, movilidad, confiabilidad o comunicación en tiempo real, pero abre nuevas oportunidades para crear e implementar soluciones innovadoras para estos y otros problemas similares. [18]

2.2.3 ARQUITECTURA SDN:

La arquitectura para las Redes Definidas por Software (SDN) es definida por la Open Networking Foundation (ONF). Esta arquitectura se fundamenta en tres principios esenciales que son:

- **Desacoplamiento de los planos de control y datos**

Las redes definidas por software (SDN) ofrecen flexibilidad a las redes actuales, lo que permite a los operadores administrar elementos de red mediante software en un servidor externo. Las SDN se basan en una característica clave: la separación del plano de control del plano de datos. Sin embargo, se entiende que ese control debe ejercerse necesariamente dentro de los sistemas del plano de datos, el protocolo OpenFlow permite la comunicación del plano de control y datos.

- **Control lógicamente centralizado**

Lógicamente centralizado significa que el control aparece desde el exterior (cliente/aplicación) en perspectiva como una sola entidad. Asegura que los recursos se pueden utilizar de manera más eficiente como en comparación con una perspectiva restringida localmente, potencialmente se puede tomar mejores

decisiones sobre cómo desplegar los recursos de red. La escalabilidad se mejora al desacoplar y centralizar el control, permitiendo vistas cada vez más globales pero menos detalladas de los recursos de la red.

- **Programabilidad de servicios de red**

Las interfaces entre los componentes de SDN exponen las abstracciones y el estado de los recursos, e intercambio de información y programación. Las aplicaciones están habilitadas para especificar requisitos y solicitar cambios en servicios de red, y para reaccionar programáticamente a los estados de la red. [19]

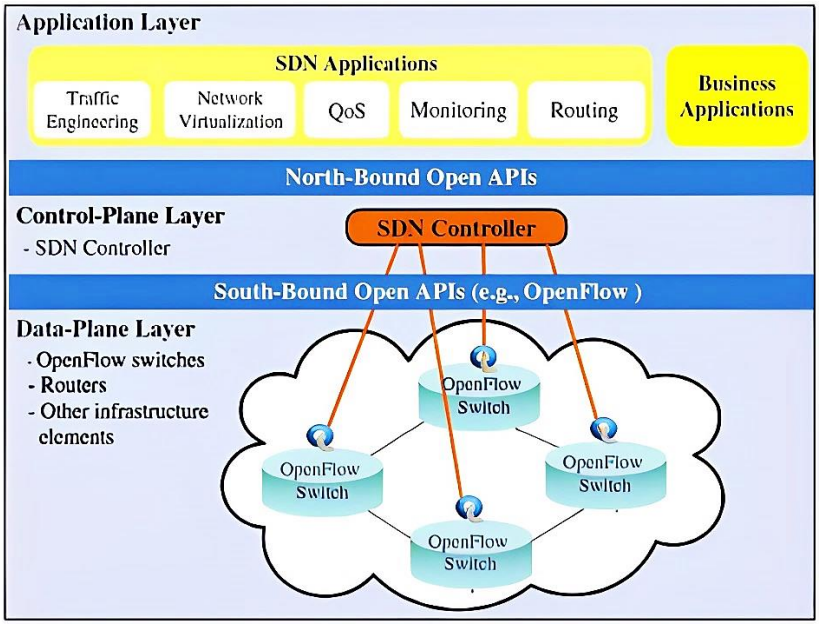


Figura 3. *Esquema de arquitectura de red SDN.*
Fuente: *Imagen extraída del sitio web ScienceDirect, A roadmap for traffic engineering in SDN-OpenFlow networks.*

Las Redes Definidas por Software (SDN) representan una tecnología emergente cuyo principio fundamental radica en la eliminación de la inteligencia inherente a la red. En su lugar, se propone la gestión de todas las funcionalidades de la red a través de un controlador centralizado. La estructura básica de una arquitectura SDN, tal como se ilustra en la Figura 3, se compone de tres planos distintos: el Plano de Aplicación, el Plano de Control y el Plano de Datos (o Infraestructura). Cada uno de estos planos desempeña un papel crucial en la operación y administración de la red.

2.2.3.1 PLANO DE DATOS:

Capa de infraestructura (también conocido como Plano de datos) en SDN se compone de dispositivos de red como router, switch y punto de acceso. Tanto conmutadores virtuales como Open vSwitch, Indigo, Pica8, Nettle, Open Flow y conmutadores físicos coexisten en esta capa. La principal función del plano de datos es reenviar los paquetes de acuerdo a reglas y políticas asignadas.

2.2.3.2 SWITCH OPENFLOW

Un switch (conmutador) en redes definidas por software es un elemento de reenvío básico que se comunica con el controlador a través de una interfaz abierta como OpenFlow. Un conmutador OF consta de al menos tres partes principales: tabla de flujo, canal seguro y protocolo OpenFlow. Las tablas de flujo se utilizan para buscar paquetes y también para reenviarlos.

Un canal seguro suele ser un canal TLS o SSL entre el conmutador y el controlador. El protocolo OpenFlow se utiliza para comunicarse con los conmutadores y gestionarlos. El controlador puede actualizar, eliminar y agregar entradas a las tablas de flujo mediante mensajes OF. Un flujo es una secuencia de paquetes que coincide con una entrada específica en una tabla de flujo. Al recibir un paquete, el conmutador hace coincidir los campos del encabezado del paquete con el componente de entradas de los campos de coincidencia. Después del proceso de búsqueda y comparación de tablas, las instrucciones de entrada con mayor prioridad se aplicarán en el paquete. [20]

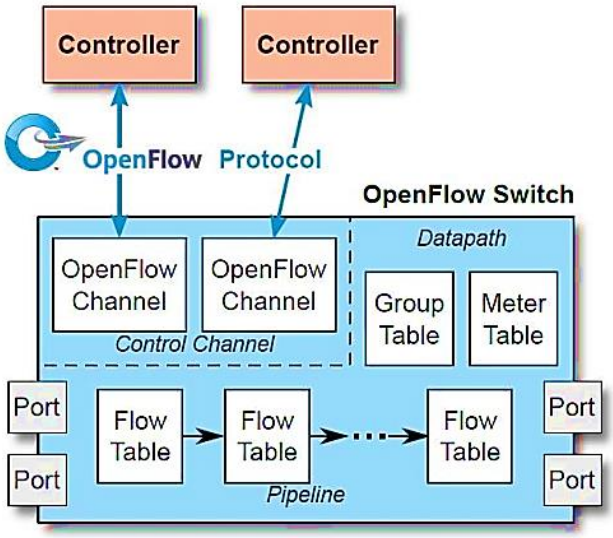


Figura 4. *Componentes de un switch OpenFlow.*
Fuente: *Imagen extraída del sitio web ONOS-Wiki, OpenFlow 1.5 Implementation.*

En otro aspecto, los conmutadores se pueden clasificar en conmutadores basados en hardware y conmutadores (virtuales) basados en software. En la tabla 1 se enumera algunos de los switch SDN disponibles.

***Tabla 1**.Switches SDN disponibles.*
***Fuente:** Información extraída de Journal of Advanced Computer Science & Technology (JACST).*

Switch	Tipo	Versión	Versión OpenFlow
SEL	Hardware	SEL-2741, SEL-2742S, SEL-2740S.	1.3
Arista	Hardware	Serie 7500R, 7200X, 7150S, 7050X, 7020R, 7300X.	1.0
HP	Hardware	3500, 3500yl, 5400zl, 6200yl, 6600, FlexFabric 12900, 5930, 5400	1.0, 1.3
NEC	Hardware	PF5240, PF5248	1.0, 1.3.1
Open vSwitch (OVS)	Software	Versiones anteriores y OVS 2.12	OpenFlow 1.0 para OVS 1.9 y anteriores, OpenFlow 1.2 y 1.3 para OVS 1.10 y posterior, OpenFlow 1.1 para OVS 2.0 y posterior, soporte experimental de OpenFlow 1.4 para OVS 2.8-2.11 Soporte de OpenFlow 1.5 para OVS 2.12
LINC	Software	-	1.0,1.1,1.2,1.3, 1.4
IBM	Hardware	IBM 8264, RackSwitch G8264, G8264T	1.0
ofsoftswitch13	Software	-	OpenFlow 1.3
Indigo Virtual Switch (IVS)	Software	Versiones anteriores e Indigo Virtual Switch 0.5	1.0, 1,1, 1.2, 1.3, 1.4, 1.5

2.2.3.3 PLANO DE CONTROL:

La capa de control en una Red Definida por Software (SDN) está compuesta por un controlador que supervisa las funciones generales de la SDN. Esta capa opera como un intermediario entre la capa de infraestructura y la capa de aplicación. El

controlador tiene la responsabilidad de administrar todo el flujo de tráfico en la red. Sus funciones incluyen la toma de decisiones sobre enrutamiento, reenvío de flujos y manejo de pérdida de paquetes, las cuales se realizan a través de programación. La comunicación entre la capa de control y la capa de infraestructura se lleva a cabo mediante un protocolo, como OpenFlow o NetConf, entre otros. Este protocolo facilita el intercambio eficiente y efectivo de información entre estas dos capas críticas en la arquitectura SDN. [21]

2.2.3.4 PLANO DE APLICACIÓN:

La capa de aplicación, tal como se ilustra en la Figura 3, representa un componente crucial en la arquitectura de las Redes Definidas por Software (SDN). Esta capa es responsable del manejo de aplicaciones comerciales y de seguridad relacionadas con el software. La virtualización de redes, los sistemas de detección de intrusos (IDS), los sistemas de prevención de intrusiones (IPS), la implementación de firewalls, las políticas de Calidad de Servicio (QoS), la ingeniería de tráfico, el enrutamiento y la gestión de la movilidad son solo algunos ejemplos de las aplicaciones gestionadas por esta capa. La comunicación entre la capa de aplicación y la capa de control se realiza a través de una interfaz de aplicación. Esta interfaz permite un intercambio eficiente y efectivo de información, facilitando así la coordinación y cooperación entre estas dos capas fundamentales en la arquitectura SDN. [22]

2.2.4 CONTROLADOR DE RED SDN:

La arquitectura de las Redes Definidas por Software (SDN) no establece especificaciones con respecto al diseño interno o la implementación de un controlador SDN. Este podría ser un único proceso monolítico, o bien, podría consistir en un conjunto de procesos organizados para compartir la carga de trabajo o protegerse mutuamente de posibles fallos. Además, el controlador SDN podría estar compuesto por distintos componentes funcionales que operan en un acuerdo de colaboración conjunta, o que se suscriben a servicios externos para desempeñar algunas de sus funciones, como por ejemplo, el cálculo de rutas.

Cualquier combinación de estas alternativas es permitida dentro de la arquitectura SDN. Desde una perspectiva externa, el controlador SDN se percibe como una “caja negra”, cuyo funcionamiento interno no es observable. Los componentes del

controlador pueden ejecutarse en plataformas informáticas arbitrarias, incluyendo recursos de cómputo locales a un elemento de red físico. También pueden operar en sistemas distribuidos y recursos migratorios, como máquinas virtuales (VM) en centros de datos. Esta flexibilidad en la implementación y operación del controlador SDN es uno de los aspectos más destacables y ventajosos de esta arquitectura. [23]

2.2.4.1 FUNCIONES BÁSICAS:

SDN separa el plano de datos y el plano de control. Esencialmente, la inteligencia de la red se implementa en el controlador de red; todos los cálculos se hacen allí y se puede añadir aplicaciones y características según sea necesario. Los controladores SDN básicos por lo general cuentan con los siguientes módulos: módulo de descubrimiento de enlaces, módulo de topología, módulo de almacenamiento, módulo de elaboración de estrategias, módulo de tabla de flujo y módulo de control de datos.

Esencialmente, dos módulos son responsables de proporcionar el servicio de enrutamiento: el módulo administrador de topología y el módulo de descubrimiento de enlaces. La función del módulo de descubrimiento de enlaces es recopilar la información del estado del enlace físico, hay dos tipos de descubrimiento de enlaces: descubrimiento de enlaces entre Nodos (conmutadores) OpenFlow y descubrimiento de enlaces entre un host final y un Nodo OpenFlow.

Esencialmente, el primero usa la capa de enlace y el protocolo de descubrimiento (LLDP). Así, la información proporcionada por el módulo de descubrimiento de enlace se usa para construir la base de datos vecina a nivel del controlador. Esta base de datos es administrada por el módulo de topología para construir la base de datos de topología global que por lo general se basa en el cálculo de la ruta más corta (y alterna) a cualquier nodo o host de OpenFlow. Cualquier cambio o ruptura de enlaces es rastreada por el módulo de descubrimiento de enlaces. Por lo tanto, el módulo de topología tiene la función de mantener la información de la topología y recalcular las rutas en la red después de cualquier modificación en la base de datos vecina. [24]

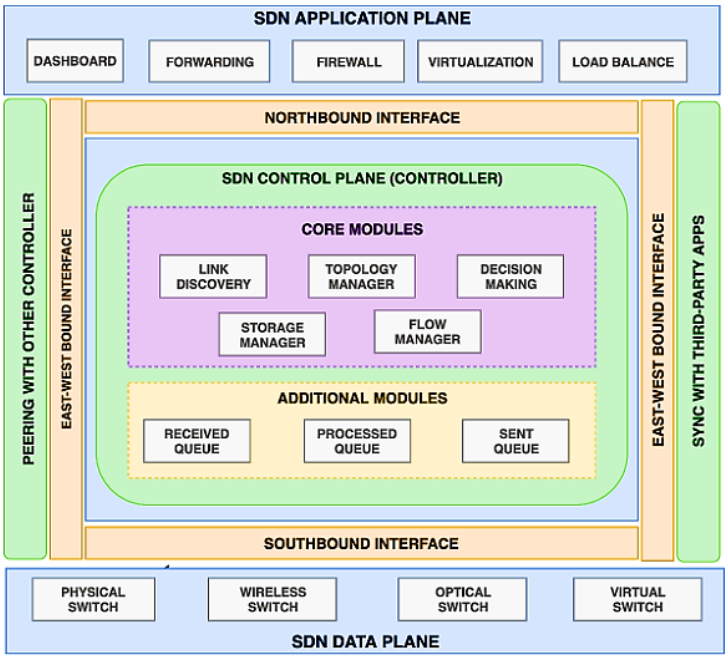


Figura 5. *Módulos de un controlador SDN.*
Fuente: *Extraído de SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study.*

INTERFACES:

El controlador central se encuentra interconectado con diversas interfaces, las cuales le permiten interactuar con otras capas y dispositivos. Esta configuración facilita la comunicación y el intercambio de información, permitiendo una gestión eficiente y efectiva de las funciones de red.

INTERFACE SOUTHBOUND (SBI):

Esta interfaz define un conjunto de reglas de procesamiento que permiten el reenvío de paquetes entre dispositivos de reenvío y controladores SBI ayuda al controlador a administrar dispositivos de red físicos y virtuales de forma inteligente. El protocolo OpenFlow (OF) es el SBI más utilizado y es un estándar por defecto para la industria. La responsabilidad fundamental de OF es comunicar el plano de datos y el plano de control, definir flujos y clasificar el tráfico de red en función de un conjunto de reglas predefinidas.

INTERFACE NORTHBOUND (NBI):

El controlador utiliza la interfaz de northbound (NBI) para permitir que los desarrolladores integren sus aplicaciones con el controlador y los dispositivos del

plano de datos. Los controladores por lo general admiten una serie de API´s orientadas al norte, pero la mayoría de ellas se basan en la API REST.

INTERFACE EAST-WEST BOUND (EWBI):

Para la comunicación entre controladores, se utiliza la interfaz West Bound (WBI). No existe una interfaz de comunicación estándar para este propósito, por lo que diferentes controladores usan diferentes mecanismos. Además, los controladores heterogéneos no suelen comunicarse entre sí. La interfaz East Bound (EBI) amplía la capacidad del controlador para interactuar con enrutadores tradicionales. BGP es el protocolo más utilizado para este propósito. [25]

2.2.4.2 LENGUAJES DE PROGRAMACIÓN:

Los controladores de Redes Definidas por Software (SDN) han sido desarrollados utilizando una variedad de lenguajes de programación, entre los que se incluyen C, C++, Java, JavaScript, Python, Ruby, Haskell y Scala. En algunos casos, se opta por desarrollar todo el controlador utilizando un único lenguaje de programación. Sin embargo, en muchos otros controladores se emplean varios lenguajes tanto en su núcleo como en sus módulos. Esta estrategia permite una asignación de memoria eficiente y la posibilidad de ejecutarse en múltiples plataformas. Más importante aún, puede resultar en un mayor rendimiento bajo ciertas condiciones. Esta diversidad en el uso de lenguajes de programación refleja la flexibilidad y adaptabilidad inherentes a la arquitectura SDN. [26]

2.2.5 PROTOCOLOS PARA REDES DEFINIDAS POR SOFTWARE (SDN):

La Open Networking Foundation (ONF) establece que el protocolo OpenFlow (OF) es reconocido como el primer protocolo de comunicaciones estándar para las Redes Definidas por Software (SDN). Adicionalmente, se identifican diversos protocolos que se utilizan en la administración de redes SDN, entre los que se incluyen: el Protocolo de Conmutación de Etiquetas Multiprotocolo (MPLS), el Protocolo de Puerta de Enlace Fronteriza (BGP), el protocolo OpFlex desarrollado por Cisco Systems, el Protocolo Extensible de Mensajería y Presencia (XMPP), el Protocolo de Configuración de Red (NETCONF) y el Protocolo de Gestión de Base de Datos Open vSwitch (OVSDB). [27]

2.2.6 OPENFLOW (OF):

OpenFlow es un framework desarrollado originalmente en la Universidad de Stanford, es reconocido como uno de los primeros estándares en la industria de las Redes Definidas por Software (SDN). En la actualidad, su administración recae en la Open Networking Foundation (ONF), una organización liderada por usuarios dedicada a los estándares abiertos y la adopción de SDN.

El protocolo de comunicación OF se definió originalmente dentro de una arquitectura SDN. Este protocolo permite que el controlador SDN interactúe directamente con el plano de reenvío de dispositivos de red, incluyendo conmutadores y enrutadores, tanto físicos como virtuales. Esta interacción posibilita la programación directa de los dispositivos de red a través de una interfaz estandarizada y robusta. Esta interfaz se adapta eficazmente a los requisitos comerciales, lo que resalta su relevancia y utilidad en el campo de las redes definidas por software. [28]

Para operar en un entorno con OpenFlow (OF), es imprescindible que cualquier dispositivo que aspire a comunicarse con un controlador de Redes Definidas por Software (SDN) sea compatible con el protocolo OpenFlow. Mediante esta interfaz, el controlador SDN tiene la capacidad de enviar modificaciones a la tabla de flujo del conmutador o enrutador. Esta funcionalidad otorga a los administradores de red la posibilidad de segmentar el tráfico, gestionar los flujos para optimizar el rendimiento y comenzar a experimentar con nuevas configuraciones y aplicaciones. Esta capacidad de adaptación y personalización resalta la importancia y la utilidad del protocolo OpenFlow.

2.2.7 ARQUITECTURA DEL PROTOCOLO OPENFLOW:

OpenFlow, en comparación con otros protocolos SDN, aprovecha eficazmente los elementos y características de hardware presentes en la mayoría de los dispositivos de red, como las tablas de enrutamiento y funciones comunes como la lectura de encabezados, el envío de paquetes a un puerto y la entrega de paquetes. Al permitir el control externo de estos elementos y funciones, OpenFlow facilita que el hardware existente admita OpenFlow mediante una actualización de firmware. Esto elimina la necesidad de un cambio completo de hardware para las empresas que buscan implementar SDN en sus productos y servicios.

La arquitectura OpenFlow propone la existencia de un controlador centralizado, un conmutador OpenFlow y un canal seguro de comunicación. Estos elementos se muestran en la Figura 6. Cada conmutador OpenFlow consta de tablas de flujo que gestiona el controlador. Cada tabla de flujo tiene tres elementos: encabezado de paquete, acciones y estadísticas. El encabezado del paquete es como una máscara que selecciona los paquetes que serán procesados por el switch. Los campos utilizados para la comparación pueden ser de la capa 2, 3 o 4 de la arquitectura TCP/IP. Eso quiere decir que no hay una separación entre capas como en las arquitecturas convencionales actuales. Todos los paquetes procesados por el conmutador se filtran a través de este método. La cantidad de campos que el conmutador puede procesar depende de la versión del protocolo OpenFlow. En OpenFlow v1.0 (la versión más utilizada), hay 12 campos, mientras que en la versión OpenFlow v1.3 define la existencia de 40 campos incluyendo soporte para IPv6. [29]

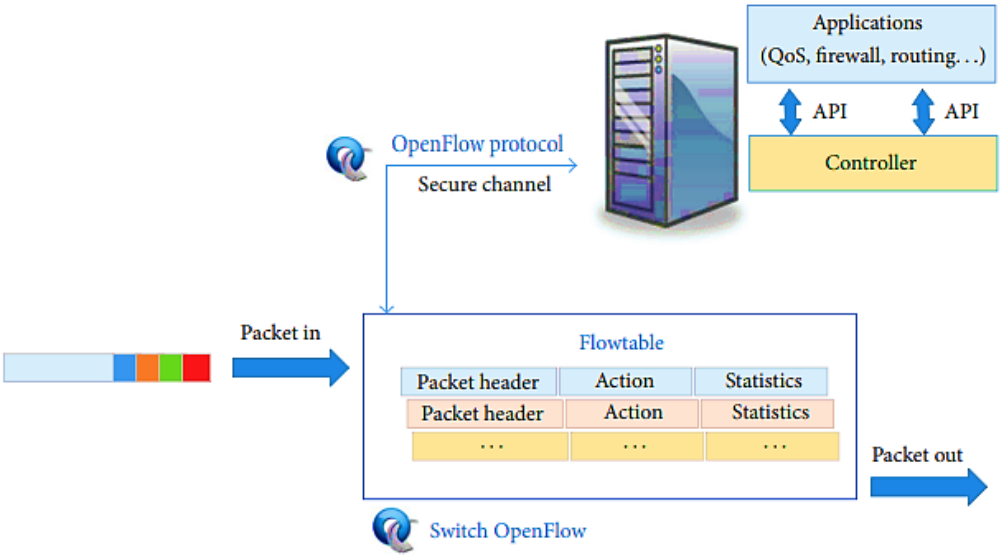


Figura 6. *Elementos de la arquitectura OpenFlow.*
Fuente: *Extraído de SDN: Evolution and Opportunities in the Development IoT Applications.*

El protocolo OpenFlow define los siguientes tipos de mensajes entre el conmutador y el controlador: controlador a conmutador, simétrico y asíncrono. Los tipos de mensajes del controlador a conmutador administra el estado del conmutador. Mensajes simétricos son enviados por el controlador al conmutador para iniciar la conexión o el intercambio de mensajes. Los mensajes asíncronos actualizan el control de los eventos de la red y los cambios de estado del

conmutador. De manera similar, OpenFlow establece dos tipos de conmutadores: OpenFlow-only y OpenFlow-enabled. Los conmutadores OpenFlow-only usan solo el protocolo OpenFlow para procesar paquetes. Por otro lado, los conmutadores OpenFlow-enabled pueden adicionalmente procesar el paquete utilizando algoritmos tradicionales de conmutación o enrutamiento. [30]

2.2.7.1 ESPECIFICACIONES PARA OPENFLOW (OF):

Cada versión del protocolo OpenFlow (OF) ha introducido especificaciones y características distintivas, junto con cambios menores y mayores en comparación con sus predecesoras. A continuación, se resaltan algunas de las características y las modificaciones implementadas en las diferentes versiones del protocolo OF.

- **OpenFlow 1.0:**

  Un conmutador habilitado para OpenFlow puede tener una o más colas según sus puertos. Un controlador OpenFlow puede consultar información sobre las colas de un conmutador. Sin embargo, el comportamiento de la cola se determina fuera del alcance de OpenFlow, y se puede configurar pero requiere OpenFlow 1.2 y versiones posteriores. Además, los campos de encabezado pueden incluir prioridad de VLAN y IP ToS, por lo que los paquetes se pueden comparar con las reglas y sus campos de encabezado asociados se pueden reescribir. [31]

- **OpenFlow 1.1:**

  Se implementa la identificación de coincidencias y etiquetado de etiquetas VLAN, MPLS y clases de tráfico. Las versiones anteriores de la especificación OpenFlow tenían compatibilidad limitada con soporte para VLAN (solo admitían un único nivel de etiquetado de VLAN con semántica ambigua). El nuevo soporte de etiquetado tiene acciones explícitas para agregar, modificar y eliminar etiquetas de VLAN, y puede admitir varios niveles de etiquetado de VLAN. Esta versión también agrega un soporte similar a los encabezados shim MPLS. [32]

- **OpenFlow 1.2:**

  Se agregó la capacidad que permite a un controlador consultar todas las colas en un conmutador. También se añadió la propiedad de cola experimental. Otra

mejora relacionada con QoS en esta versión es que ha agregado una propiedad de tasa máxima de cola. Además, esta versión especifica que las colas se pueden ligar a los puertos y usarse para mapear flujos en ellos. [33]

- **OpenFlow 1.3:**

  Se introduce la funcionalidad de limitación de tasa (rate-limiting) que consisten en medir las entradas por medio de medidores de tablas. Los contadores ayudan al controlador a recopilar estadísticas sobre la red, los medidores se pueden combinar con la acción opcional de establecer cola, que asocia un paquete a una cola y un puerto en jerarquía para implementar marcos de QoS complejos. Estos medidores complementan el framework de colas que ya existe en OpenFlow al permitir el monitoreo de la tasa del tráfico antes de la salida. Más específicamente, con medidores, podemos monitorear la tasa de ingreso de tráfico definida por una regla de flujo. Los paquetes se pueden dirigir a un medidor específico mediante la instrucción opcional de medidor (meter id), donde el medidor puede realizar algunas operaciones en función de la tasa a la que recibe los paquetes. [34]

- **OpenFlow 1.4:**

  Se presenta el framework de monitoreo de flujo que permite a un controlador monitorear los cambios realizados por otros controladores en cualquier subconjunto de las tablas de flujo en tiempo real. Con este fin, un controlador puede definir un número de monitores, seleccionando un subconjunto de las tablas de flujo. Cada monitor incluye una identificación de tabla (table id) y un patrón de coincidencia que define el subconjunto monitoreado. Cuando se agrega, modifica o elimina cualquier entrada de flujo en uno de los subconjuntos definidos por un monitor de flujo, se envía un evento al controlador para informarle sobre el cambio realizado. [35]

- **OpenFlow 1.5:**

  Se reemplaza la instrucción meter, que se usaba para medición en versiones anteriores, con una acción de medidor (meter action). Como resultado, se pueden agrupar varios medidores a una entrada de flujo y los medidores se pueden usar en segmentos grupales. [36]

Tabla 2. *Especificaciones del protocolo OpenFlow.*
Fuente: *Extraído de Open Networking Foundation, OpenFlow Switch Specification.*

Versión	Característica Principal	Razón	Casos de Uso
1.0 - 1.1	Tabla multiple	Evita la explosión de entrada de flujo	
	Tabla de Grupo	Habilitar la aplicación de conjuntos de acciones a un grupo de flujos	Equilibrio de carga, tolerancia frente a fallos, agregación de enlaces
	VLAN y Soporte MPLS		
1.1 - 1.2	OpenFlow Extensible Match (OXM)	Se amplía la flexibilidad de coincidencia	
	Controlador Múltiple	Equilibrio de carga/escalabilidad	Controlador con tolerancia frente a fallos, equilibrio de carga de controlador
1.2 - 1.3	Medidor de tabla	Añade QoS y Capacidad DiffServ.	
	Omisión de entrada de tabla	Proporcionar flexibilidad.	
1.3 - 1.4	Sincronización de tabla	Mejorar la escalabilidad de tabla.	Aprendizaje de MAC, reenvio de tráfico (Forwarding).
	Secuencia de solicitudes OpenFlow (Bundle)	Mejora la sincronización de switch.	Configuración de múltiples switch.
1.4 - 1.5	Tabla de salida	Permitir que el procesamiento se realice en el puerto de salida.	
	Programación de secuencia de solicitudes OpenFlow (Bundle)	Mejorar aún más la sincronización de switch.	

[37]

2.2.8 PLATAFORMAS DE CONTROLADORES SDN:

El propósito inicial de la Red Definida por Software (SDN) era centralizar el plano de control, lo que llevó a que la mayoría de los controladores adoptaran una arquitectura de despliegue centralizada compuesta por un solo controlador. Sin

embargo, esto generó un único punto de fallo y desafíos de escalabilidad. La arquitectura distribuida permite el uso de múltiples controladores dentro de un dominio, trabajando en formación conjunta o jerárquica. Aunque existen múltiples controladores SDN, su funcionamiento es más o menos el mismo en todos ellos. Las 20 propuestas enumeradas en la Tabla 3 ofrecen una visión global y detallada de las características de cada controlador SDN.

Tabla 3. *Diferentes plataformas de controladores SDN.*
Fuente: *Extraído de IEEE Xplore, SDN Controllers: A Comparative Study.*

Controlador	Interfaz	Lenguaje	Plataforma Soportada	Arquitectura	Southbound Api	Northbound Api	Eastwestbound Api
NOX	Web, CLI	C++	Linux	Centralizado	OpenFlow 1.0	ad-hoc	-
POX	GUI, CLI	Python	Windows Linux, MacOS.	Centralizado	OpenFlow 1.0	ad-hoc	-
Floodlight	Web, CLI	Java	Windows Linux, MacOS.	Centralizado	OpenFlow 1.0, 1.3	Quantum, REST, Java RPC.	-
OpenDaylight	Web, CLI	Java	Windows Linux, MacOS.	Distribuido	OpenFlow 1.0, 1.3	NETCONF, REST, RESTCONF, XMPP.	Raft, Akka.
ONOS	Web, CLI	Java	Windows Linux, MacOS.	Distribuido	OpenFlow 1.0, 1.3	Neutron, REST.	Raft
Ryu	CLI	Python	Linux, MacOS.	Centralizado	OpenFlow 1.0-1.5	REST	-
Beacon	Web, CLI	Java	Windows Linux, MacOS.	Centralizado	OpenFlow 1.0	ad-hoc	-
Trema	CLI	Ruby, C.	Linux	Centralizado	OpenFlow 1.0	ad-hoc	-
RunOS	Web, CLI	C++	Linux	Distribuido	OpenFlow 1.3	REST	Maple

OpenContrail	Web, CLI	Python, C++, C.	Linux	Centralizado	REST	XMPP, BGP.	-
Microflow	Web, CLI	C	Linux	Centralizado	OpenFlow 1.0-1.5	Socket	-
McNettle	CLI	Haskell	Linux	Centralizado	OpenFlow 1.0	-	-
Maestro	Web, CLI	Java	Windows Linux, MacOS.	Centralizado	OpenFlow 1.0	ad-hoc	-
NodeFlow	CLI	JavaScript	Node.js	Centralizado	OpenFlow 1.0	JSON	-
Onix	-	C++	-	Distribuido	OVSDB, OpenFlow 1.0	Onix API	Zookeeper
FlowVisor	CLI	C	Linux	Centralizado	OpenFlow 1.0, 1.3	JSON RPC	-
Faucet	Web, CLI	Python	Linux	Centralizado	OpenFlow 1.3	-	-
ZeroSDN	Web, CLI	C++	Linux	Distribuido	OpenFlow 1.0, 1.3	REST	ZeroMQ
Ravel	CLI	Python	Linux	Centralizado	OpenFlow 1.0	ad-hoc	-
TinySDN	CLI	C	Linux	Centralizado	OpenFlow 1.0	-	-

2.2.9 SIMULADORES Y EMULADORES PARA REDES DEFINIDAS POR SOFTWARE (SDN):

Los emuladores y simuladores de redes de Definidas por Software (SDN) proporcionan una plataforma invaluable para que los investigadores y profesionales de redes puedan evaluar el comportamiento de las redes bajo cargas de trabajo específicas. Con la introducción del protocolo OpenFlow, se han realizado mejoras significativas en los simuladores existentes, incorporando componentes adicionales para soportar escenarios basados en OpenFlow. Paralelamente, se han desarrollado numerosos emuladores habilitados para SDN

que se basan en conmutadores de software, como Open vSwitch (OvS), ofsoftswitch13 o Indigo Virtual Switch (IVS). [38]

La tabla que se muestra a continuación ofrece una comparación detallada entre varios simuladores y emuladores que se utilizan en el ámbito de las redes de Definidas por Software (SDN).

Tabla 4. *Comparativa de diferentes simuladores/emuladores SDN.*
Fuente: *Extraído de A Comprehensive Survey on Management Tools and Techniques.*

Herramienta	Descripción	Interfaz	Tipo	Escalabilidad
Mininet	Emulador de red SDN basado en Open vSwitch	OpenFlow 1.0,1.2,1.3, 1.4, 1.5	Emulador	Mediana escala
MaxiNet	Emulador de red SDN a gran escala basado en Mininet	OpenFlow 1.0,1.2,1.3, 1.4, 1.5	Emulador	Gran escala
EstiNet	Simulador y emulador OpenFlow	OpenFlow 1.0,1.1,1.3, 1.4	Simulador/Emulador	Gran escala
ViNo	Orquestador de redes virtuales enfocado en la migración de máquinas virtuales.	-	Emulador	-
OFNet	Emulador SDN que incluye generador de tráfico.	OpenFlow (Versión no especificada)	Emulador	-
DOT	Emulador de red SDN a gran escala basado en Open vSwitch	OpenFlow 1.0,1.3	Emulador	-
OMNeT++	Simulador de eventos discretos con soporte OpenFlow	OpenFlow 1.0	Simulador	Gran escala
fs-sdn	Extensión del simulador fs	POX NBI	Simulador	Mediana escala
Ns-3	Simulador de eventos discretos con soporte OpenFlow	OpenFlow 0.8.9, 1.3	Simulador	Gran escala

2.2.10 VIDEO STREAMING:

El término “streaming” se refiere a la transmisión continua de archivos multimedia desde un servidor a un cliente en tiempo real mediante un conjunto de protocolos. Este proceso comienza con un archivo multimedia pregrabado alojado en un servidor remoto. A diferencia de las descargas tradicionales, el contenido se transmite en línea sin necesidad de descargar el archivo completo. Al recibir una solicitud del cliente, los datos del archivo se comprimen y se envían en partes al dispositivo solicitante. Estos medios se reproducen a medida que llegan, utilizando un reproductor multimedia especializado que descomprime y envía los datos a la pantalla y los altavoces. Los archivos se eliminan automáticamente una vez reproducidos. Ejemplos de reproductores multimedia incluyen VLC, Windows Media Player y QuickTime Player. Hoy en día, el streaming de vídeo domina el tráfico de Internet, impulsado por el crecimiento exponencial de las plataformas de redes sociales. [39]

La transmisión de video es posible gracias a protocolos y tecnologías especializados que permiten dividir el contenido en pequeños paquetes de datos y transmitirlos en un flujo continuo. Se utiliza un protocolo de transmisión, la mayor parte del tiempo el Protocolo de control de transmisión (TCP) o el Protocolo de datagramas de usuario (UDP), para intercambiar datos a través de una red. Algunas de las plataformas de transmisión más conocidas incluyen Netflix, Facebook, Instagram, YouTube, Amazon Prime Video, Disney+, Hulu, Twitch y TikTok. Estas plataformas albergan una amplia variedad de contenidos, que atienden a los diferentes intereses y preferencias de los espectadores.

2.2.11 PROTOCOLOS DE TRANSPORTE TCP/UDP:

PROTOCOLO DE CONTROL DE TRANSMISIÓN (TCP/IP):

El Protocolo de Control de Transmisión (TCP/IP) es un protocolo de transporte robusto orientado a la conexión, que facilita el intercambio de paquetes de datos a través de la red. TCP actúa como el regulador del tráfico en la autopista digital, asegurando que los dispositivos de red, como los enrutadores y conmutadores, entreguen paquetes de datos de manera precisa, confiable y en la secuencia correcta. Proporciona un servicio confiable mediante el establecimiento de conexión, la detección de errores, la retransmisión y el control de congestión. TCP

garantiza una comunicación exitosa entre los dispositivos y luego supervisa y ajusta constantemente la velocidad de transmisión. Asegura una transferencia de datos eficiente y confiable, incluso en situaciones de congestión de la red y pérdida de paquetes. Esta gestión inteligente del tráfico digital ha sido fundamental para el éxito y la escalabilidad de Internet tal como la conocemos hoy en día. [40]

PROTOCOLO DE DATAGRAMAS DE USUARIO (UDP):

El Protocolo de Datagramas de Usuario (UDP, por sus siglas en inglés) es un protocolo de comunicaciones que se utiliza principalmente para establecer conexiones de baja latencia y tolerancia a pérdidas entre aplicaciones en Internet. Se considera un protocolo sin conexión debido a que no requiere el establecimiento de un circuito virtual antes de que se produzca cualquier transferencia de datos. UDP acelera las transmisiones al permitir la transferencia de datos antes de que la parte receptora proporcione un acuerdo. Como resultado, UDP no proporciona una comunicación confiable y es beneficioso en comunicaciones urgentes y aplicaciones que requieran un gran ancho de banda, como el streaming de video, los juegos en línea y la multidifusión. La confiabilidad no es la característica técnica requerida que se espera de aplicaciones como el streaming de audio/video, video bajo demanda, etc. La mayoría de estas aplicaciones utilizan UDP como protocolo de transporte. Sin embargo, UDP no tiene ningún mecanismo de control de congestión, lo que puede provocar inestabilidad en Internet. [41]

El uso de UDP puede representar una amenaza significativa para la estabilidad de Internet. Al principio, este problema no se consideró seriamente debido a que las aplicaciones que utilizaban UDP eran principalmente del tipo solicitud-respuesta. Las aplicaciones como el Servicio de Nombres de Dominio (DNS) transmiten únicamente mensajes cortos. Sin embargo, con el uso de UDP para aplicaciones de streaming, esto puede afectar considerablemente el rendimiento de Internet. La razón principal por la que estas aplicaciones prefieren UDP a TCP es la menor sobrecarga asociada con UDP. Estas aplicaciones no requieren la retransmisión de paquetes, ya que en las aplicaciones de streaming un paquete retrasado es equivalente a la pérdida de un paquete. Por lo tanto, existe la necesidad de un

protocolo de transporte que admita el control de la congestión pero evite la sobrecarga asociada con la retransmisión. [42]

El Protocolo de Control de Transmisión (TCP) se distingue por su fiabilidad, mientras que el Protocolo de Datagramas de Usuario (UDP) es reconocido por su rapidez en la transmisión. UDP se emplea cuando la velocidad es una prioridad sobre la confiabilidad, y TCP se utiliza cuando la confiabilidad es de mayor importancia. Los servicios de streaming para consumidores suelen utilizar TCP, mientras que UDP es ideal para videoconferencias y aplicaciones de multidifusión.

2.2.12 PROTOCOLOS DE APLICACIÓN PARA STREAMING:

Existen varios protocolos destacados para la transmisión de streaming (video y audio) a través de una red, entre los cuales se incluyen los siguientes:

2.2.12.1 PROTOCOLO RTSP:

El Protocolo de Streaming en Tiempo Real (RTSP, por sus siglas en inglés) es un protocolo de nivel de aplicación diseñado para el control de la entrega de datos que poseen propiedades en tiempo real. RTSP ofrece un marco de trabajo extensible que facilita la entrega controlada y a demanda de datos en tiempo real, tales como audio y video. Las fuentes de estos datos pueden comprender tanto transmisiones de datos en vivo como clips previamente almacenados. Este protocolo está diseñado para controlar múltiples sesiones de entrega de datos, proporcionando un medio para seleccionar canales de entrega como UDP, multidifusión UDP y TCP. Además, ofrece un medio para elegir mecanismos de entrega basados en RTP. En este sentido, RTSP actúa como un “control remoto de red” para servidores multimedia. [43]

Es importante destacar que no existe la noción de conexión RTSP; en su lugar, un servidor mantiene una sesión identificada por un identificador único. Una sesión RTSP no está vinculada de ninguna manera a una conexión a nivel de transporte, como podría ser una conexión TCP.

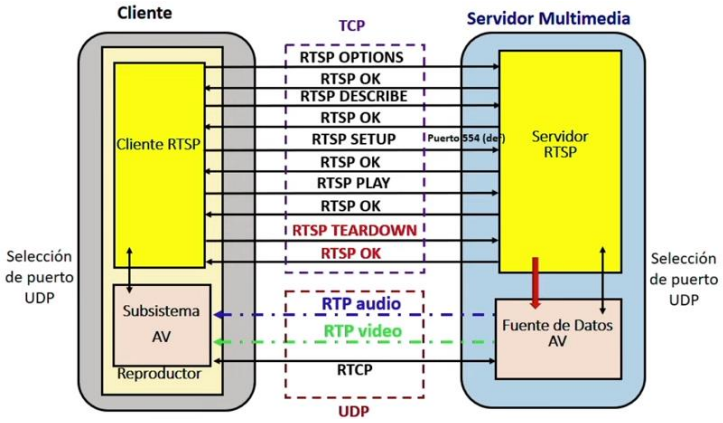


Figura 7. *Protocolo de streaming en tiempo real (RTSP).*
Fuente: *Extraído de Universidad Politécnica de Valencia: Análisis del protocolo RTSP con Wireshark.*

2.2.12.2 PROTOCOLO RTMP:

El Protocolo de Mensajería en Tiempo Real (RTMP) de Adobe, es un protocolo de nivel de aplicación que ha sido diseñado para proporcionar un servicio multiplex de mensajes bidireccionales a través de un transporte de flujo confiable. Este protocolo está destinado a transportar flujos paralelos de mensajes de video, audio y datos, todos ellos con información temporal asociada entre dos puntos comunicantes. Las implementaciones de este protocolo suelen asignar diferentes prioridades a diferentes clases de mensajes. Esta asignación de prioridades puede afectar el orden en el que los mensajes se ponen en cola en el transporte de flujo subyacente cuando la capacidad del transporte está limitada. En otras palabras, cuando la capacidad del transporte está restringida, la prioridad asignada a cada clase de mensaje puede determinar el orden en que estos mensajes son procesados. [44]

Figura 8. *Aplicación actual del protocolo RTMP en la transmisión de streaming.*
Fuente: *Extraído de RTMP vs RTSP, MarcSoul.*

2.2.12.3 PROTOCOLO RTP:

El protocolo de transporte en tiempo real RTP proporciona servicios de entrega de extremo a extremo para datos con características en tiempo real, como audio y vídeo interactivos o datos, a través de servicios de red de multidifusión o unidifusión. RTP no aborda la reserva de recursos y no garantiza la calidad del servicio para los servicios en tiempo real. RTP está diseñado para ser independiente de las capas de red y transporte subyacentes. Las aplicaciones normalmente ejecutan RTP sobre UDP para utilizar sus servicios de multiplexación y suma de comprobación; Ambos protocolos contribuyen con partes de la funcionalidad del protocolo de transporte. Sin embargo, RTP se puede utilizar con otros protocolos de transporte o de red subyacentes adecuados. [45]

RTP admite la transferencia de datos a múltiples destinos mediante distribución de multidifusión si la proporciona la red subyacente. RTP en sí no proporciona ningún mecanismo para garantizar la entrega oportuna ni ofrece otras garantías de calidad de servicio, sino que depende de servicios de capa inferior para hacerlo. No garantiza la entrega ni evita la entrega fuera de servicio, ni asume que la red subyacente sea confiable. Los números de secuencia incluidos en RTP permiten al receptor reconstruir la secuencia de paquetes del remitente, pero los números de secuencia también pueden usarse para determinar la ubicación adecuada de un paquete, por ejemplo en la decodificación de video, sin necesariamente decodificar los paquetes en secuencia. [46]

Tabla 5. *Protocolos destinados a la transmisión de streaming.*
Fuente: *Elaborado por el autor.*

RTSP	RTMP	RTP
Codecs H.264, H.265, VP8, VP9.	Codecs H.264, VP6, VP8, Screen Video v1, v2.	H.264, H.265, VP8, VP9 MP4V-ES, MPEG-4.
Menos popular	Estabilidad y popularidad	Estabilidad
Segmentación integrada en sintaxis	Sin segmentación	Segmentación integrada en sintaxis
Latencia 2 seg.	Latencia 3-30 seg.	Latencia 2 seg.
Puerto por defecto 554 TCP y UDP.	Puerto por defecto 1935 TCP.	Puertos entre 1024 – 65535 (por lo general 5004).

2.2.13 TIPOS DE TRANSMISIÓN:

2.2.13.1 TRANSMISIÓN UNICAST:

Unicast es el término utilizado para describir la comunicación en la que se envía una información de un punto a otro. En este caso solo hay un remitente y un receptor. La transmisión Unicast, en la que se envía un paquete desde una única fuente a un destino específico, sigue siendo la forma predominante de transmisión en las redes LAN y en Internet. Todas las redes LAN y redes IP admiten el modo de transferencia en unidifusión, y la mayoría de los usuarios están familiarizados con las aplicaciones de unidifusión estándar que emplean el protocolo de transporte TCP. [47]

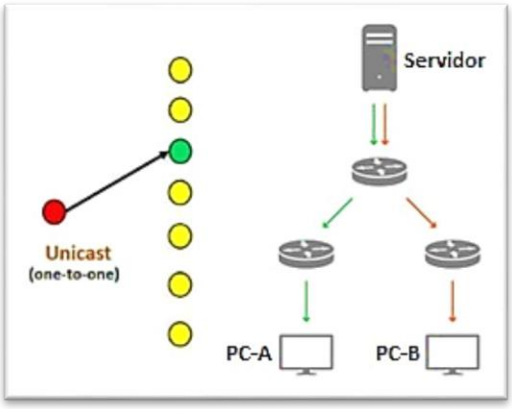


Figura 9. *Comunicación en Unicast.*
Fuente: *Extraído del Sitio web TV y Video.*

2.2.13.2 TRANSMISIÓN BROADCAST:

Es el proceso en el cual un nodo emisor transmite un paquete de datos a todos los demás nodos en una red. La transmisión en toda la red se denomina simplemente "difusión". La transmisión implica el envío de una señal desde una única fuente a múltiples receptores de manera simultánea, permitiendo que cualquier dispositivo receptor dentro del alcance de la red pueda visualizar el contenido transmitido. Este método es ampliamente utilizado en la televisión para la distribución de contenido en tiempo real. La principal ventaja de este método de transmisión radica en su capacidad para llegar a audiencias a gran escala, funcionando de manera análoga a una difusión única para todos los receptores. [48]

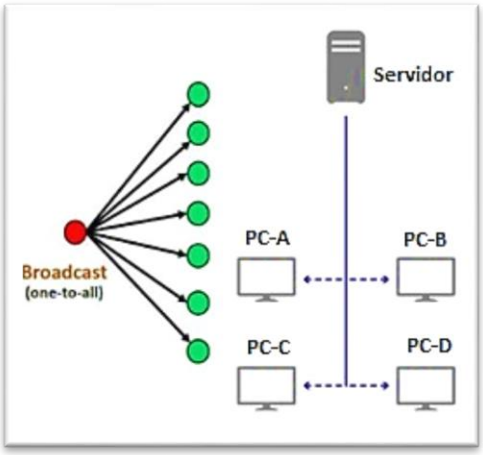


Figura 10. *Comunicación en Broadcast.*
Fuente: *Extraído del Sitio web TV y Video.*

2.2.13.3 TRANSMISIÓN MULTICAST:

La transmisión multicast de uno a muchos es una tecnología que facilita la comunicación entre un conjunto de computadoras que comparten una dirección de multidifusión. Se caracteriza por el envío de información desde una única fuente a múltiples destinatarios en una red, donde el receptor emite una señal para aceptar la transmisión. El esquema de direccionamiento IP clase D de multicast es administrado y asignado por la Autoridad de Números Asignados de Internet (IANA). Las direcciones de clase E están reservadas para futuras necesidades de direccionamiento. Mediante el uso del protocolo IGMP y los protocolos de enrutamiento de multidifusión, se puede configurar una red con subredes para utilizar y reenviar paquetes de multidifusión. Los hosts pueden informar su incorporación o salida de un grupo mediante IGMP. La multidifusión reduce significativamente el tráfico de la red al entregar una única secuencia de video a múltiples receptores. [49]

La multidifusión, a diferencia de otras técnicas, se diseñó para transmitir audio y vídeo en streaming a través de redes IP, permitiendo la entrega de una secuencia de vídeo a múltiples destinatarios. Establece una conexión unidireccional, por lo que no puede ser transportada a través del protocolo TCP, que requiere confirmación de conectividad. Por su naturaleza, la multidifusión se envía a través del Protocolo de Datagramas de Usuario (UDP) y requiere que los puertos estén abiertos.

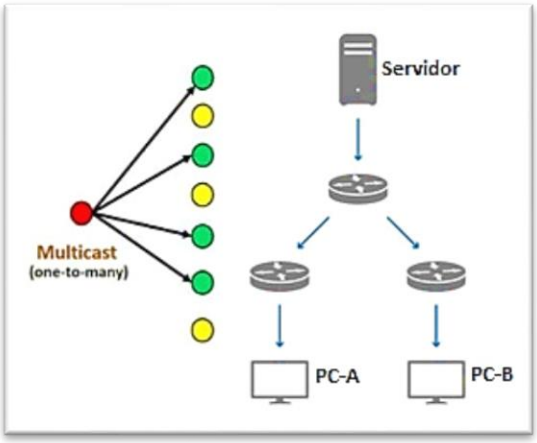


Figura 11. *Comunicación en Multicast.*

Fuente: *Extraído del Sitio web TV y Video.*

2.2.14 IP MULTICAST:

El estándar de redes IPv4 designa las direcciones IP de clase D como reservadas para multidifusión. El rango completo de todas las direcciones IP para multidifusión se muestra en la Tabla 6. La multidifusión es un mecanismo que permite definir grupos de nodos y enviar mensajes IP a dicho grupo en lugar de a cada nodo en la red LAN (difusión) o simplemente a otro nodo (unidifusión). Aunque algunas direcciones, como “224.0.0.1” para todos los hosts y “224.0.0.2” para todos los enrutadores, en la red local están definidas en el protocolo, la Autoridad de Números Asignados de Internet (IANA) es la entidad que asigna otras direcciones reservadas para multidifusión. Este rango único de direcciones IP se utiliza exclusivamente para la dirección de destino del tráfico de multidifusión IP, ya que el servidor, codificador o cámara IP que envía los datagramas de multidifusión siempre utiliza una dirección IP de origen de unidifusión. [50]

Tabla 6. *Clases de direcciones IP.*
Fuente: *Extraído del sitio web Paessler, Dirección IP.*

Clase	Rango de direcciones (Primer Octeto)		Primer Bit	Bits de red	Bits de Host	Bits Multicast	Número de redes	Número máximo de host
Clase A	0.0.0.0	127.255.255.255	0	7	24	N/A	126 (0 y 127 son	16,777.214

							reservadas)	
Clase B	128.0.0.0	191.255.255.255	10	14	16	N/A	16.384	65.532
Clase C	192.0.0.0	223.255.255.255	110	21	8	N/A	2,097.152	254
Clase D	224.0.0.0	239.255.255.255	1110	N/A	N/A	128	RFC 1112	-

El rango de tránsito de los paquetes IP está definido por el campo TTL (Time to Live), que se ubica en la cabecera del paquete. Este campo opera como un mecanismo de seguimiento de los saltos realizados por el paquete, estableciendo así el límite de la red que el paquete puede atravesar. Al inicio, se asigna un valor TTL específico dentro de la aplicación. Con cada salto que realiza el paquete, este valor TTL se reduce en una unidad. Cuando el valor TTL alcanza cero, resulta en la pérdida del paquete, dado que se ha excedido el número máximo de saltos permitidos para llegar a su destino. Comúnmente, en una red local para comunicaciones multicast, se configura un valor TTL de 1. [51]

2.2.15 PROTOCOLO DE GESTIÓN DE GRUPOS (IGMP):

El Protocolo de Gestión de Grupos de Internet (IGMP, por sus siglas en inglés) es un protocolo diseñado para proporcionar servicios de multidifusión a los hosts finales en IPv4. IGMP se emplea entre los hosts y su enrutador de multidifusión local para establecer y mantener membresías en grupos de multidifusión. IGMP se transmite directamente en un paquete IP, con un número de protocolo IP fijado en 2. IGMP opera generalmente en un entorno de subred donde todos los hosts están conectados a un enrutador perimetral. Cuando un paquete de multidifusión para un grupo de multidifusión específico llega a un enrutador perimetral, este utiliza la dirección de multidifusión IPv4 224.0.0.1, junto con el campo de Tiempo de Vida (TTL) establecido en 1, para informar a todos los hosts de la subred. Este método se asemeja más a un modo de empuje (push). La multidifusión por Internet también permite un modo de extracción (pull). Por ejemplo, si un usuario se entera de una sesión de multidifusión al leer una página web que enumera una dirección de multidifusión global específica para una presentación en vivo, la máquina host

del usuario enviaría una solicitud para unirse al grupo. IGMP cuenta con las versiones IGMP v1, v2 y v3. [52]

2.2.15.1 IGMPv1:

Los anfitriones tienen la capacidad de unirse a grupos de multidifusión. No existen mensajes para abandonar el grupo (leave group). Los enrutadores emplean un mecanismo basado en tiempo de espera para identificar los grupos que no son de interés para los miembros. [53]

2.2.15.2 IGMPv2:

Se han incorporado mensajes de abandono al protocolo (leave group), lo que facilita la notificación rápida de la finalización de la membresía del grupo al protocolo de enrutamiento. Esto resulta crucial para grupos de multidifusión con un gran ancho de banda y/o subredes con una membresía de grupo altamente volátil. [54]

2.2.15.3 IGMPv3:

Una revisión significativa del protocolo permite a los hosts especificar la lista de hosts de los que desean recibir tráfico. El tráfico proveniente de otros hosts se bloquea dentro de la red. Además, este protocolo permite a los hosts bloquear dentro de la red los paquetes que provienen de fuentes que envían tráfico no deseado. [55]

Esencialmente, existen 5 tipos de mensajes que deben ser implementados para asegurar el correcto funcionamiento de IGMPv3 y su compatibilidad con versiones anteriores:

Tabla 7. *Clases de mensajes del protocolo IGMP.*
Fuente: *Extraído de RFC 3376.*

Hexadecimal	Versión	Tipo de Mensaje	Descripción
0x11		membership query	Las consultas de membresía son emitidas por los enrutadores de multidifusión IP con el objetivo de indagar sobre el estado de recepción de multidifusión de las interfaces adyacentes.

0x22	versión 3	membership report	Los sistemas IP emiten informes de membresía de la versión 3 con el propósito de notificar a los enrutadores vecinos acerca del estado actual de recepción de multidifusión, o de los cambios en dicho estado, en sus interfaces.
0x12	versión 1	membership report	Este mensaje indica que un host tiene interés en un grupo de multidifusión específico. IGMPv1 es una versión inicial del protocolo y presenta una estructura de mensaje más sencilla en comparación con las versiones posteriores.
0x16	versión 2	membership report	En IGMPv2, los hosts utilizan el mensaje "version2 membership report" con el objetivo de indicar su interés en recibir tráfico de multidifusión de un grupo de multidifusión específico. Este mensaje se transmite a intervalos regulares para informar a los enrutadores que el host aún mantiene su interés en permanecer como miembro del grupo de multidifusión.
0x17	versión 2	leave group	Ocurre cuando el anfitrión toma la decisión de retirarse del grupo de multidifusión.

2.2.16 MEDIDAS DE RENDIMIENTO DE RED

Para llevar a cabo una evaluación efectiva del rendimiento de una red SDN, resulta esencial resaltar los siguientes conceptos fundamentales en relación con la calidad del servicio.

Tabla 8. *Medidas de rendimiento de red.*

Fuente: *Elaborado por el Autor.*

Parámetro	Descripción
Ancho de Banda (BW)	Se describe como la capacidad máxima de transferencia de datos de un medio de transmisión en un intervalo de tiempo específico. Esta capacidad se mide comúnmente en términos de bits por segundo (bps).
Throughput	Es un indicador clave del rendimiento de una red, denota la cantidad de datos que pueden transmitirse de un lugar a otro en un período de tiempo específico. Se utiliza para cuantificar

	la velocidad de transmisión de datos y se mide comúnmente en bits por segundo (bps).
Latencia	La latencia, también conocida como retraso, se define como el intervalo de tiempo que demora la información en desplazarse desde su punto de origen hasta su destino a través de una red. Este factor influye directamente en la velocidad de transmisión de los datos entre dos puntos y se mide en milisegundos (ms).
Jitter	El "Jitter" se define como la variación en la latencia de una red. Este término hace referencia a las fluctuaciones temporales que experimentan los paquetes de datos en su tránsito desde el origen hasta el destino. En esencia, el jitter representa la variabilidad en el retraso de los paquetes de datos dentro de una red. Es una métrica crucial para evaluar la fiabilidad de una red. Se cuantifica en unidades de milisegundos (ms).
Packet Loss	Se refiere a la situación en la que uno o más paquetes de datos no llegan a su destino previsto. Este fenómeno puede ser atribuido a diversas causas, entre las que se incluyen la congestión de la red, los errores durante la transmisión, la corrupción de los datos y las fallas en el hardware o el software.

2.3 MARCO TEÓRICO

Se presentan indagaciones previas que han desempeñado un papel fundamental al proporcionar una base directriz para el abordaje y el desarrollo del problema de investigación en estudio. Estas investigaciones preexistentes, permiten una comprensión más profunda y un enfoque preciso hacia los aspectos fundamentales del análisis y la resolución de la problemática planteada.

"Estudio sobre las Redes Definidas por Software (SDN) para mejorar la aplicación del Streaming de video en las Redes de Distribución de Contenidos (CDN) en el Ecuador"; El presente proyecto tiene como objetivo llevar a cabo una evaluación exhaustiva de la coyuntura actual del país en lo que respecta al streaming de video. Esta evaluación se llevada a cabo a través de la implementación de simulaciones que modelan el comportamiento de una red CDN (Red de entrega de contenido), empleando tecnologías de redes definidas por software (SDN). El propósito primordial radica en analizar el rendimiento de esta red optimizada y su sinergia con SDN. En esencia, el núcleo del estudio se enfoca en evaluar el impacto del streaming de video en la infraestructura de red, a través de un análisis minucioso

que abarca desde la ejecución de simulaciones hasta la implementación estratégica de tecnologías SDN para optimizar la gestión del tráfico multimedia. [56]

“Diseño y simulación de un prototipo de red definida por software (SDN) usando el protocolo OpenFlow”; El propósito de este estudio es la concepción y desarrollo de un prototipo de red definida por software, aprovechando herramientas de virtualización reconocidas. Esta iniciativa busca ilustrar tanto los fundamentos teóricos como los aspectos prácticos inherentes a esta arquitectura innovadora, fundamentada en el protocolo Openflow 1.5. Además, persigue el objetivo de evaluar las ventajas y desventajas de los componentes que conforman estas redes, así como su aplicabilidad en contextos empresariales. A través de estas exploraciones, se procura proporcionar a los administradores de redes una perspectiva actualizada acerca de esta tecnología en constante evolución. [57]

“Sistema de distribución Multicast mediante redes definidas por software”; El enfoque de este proyecto recae en redes LAN y LAN-Campus, con el propósito de optimizar la administración del tráfico y la transmisión multicast. Específicamente, se prioriza aplicaciones como la transmisión de contenido multimedia, para la cual se ha concebido un controlador que gestiona el flujo de tráfico multicast y utiliza paquetes IGMP para configurar de manera adecuada dichos flujos. Adicionalmente, se implementa el algoritmo de Dijkstra para establecer rutas eficientes entre nodos. El desarrollo del proyecto emplea herramientas de código abierto tales como RYU, OpenFlow, Mininet y VLC, mientras que Python se encarga de automatizar las simulaciones, ejecutar pruebas y estructurar los datos. [58]

CAPÍTULO III

3.1 COMPONENTES DE LA PROPUESTA

3.1.1 COMPONENTES LÓGICOS

Para la implementación de los escenarios de simulación y para facilitar el avance de la investigación, se emplearon los siguientes componentes de software. Estos componentes han sido fundamentales para la ejecución exitosa de la propuesta tecnológica.

3.1.1.1 SISTEMA OPERATIVO (OS):

La elección del sistema operativo es crucial para garantizar una experiencia de alta productividad durante el desarrollo e investigación. En este sentido, se seleccionó el Sistema Operativo Ubuntu 18.04.6 LTS debido a su óptima combinación de estabilidad, seguridad, eficiencia, amplios recursos de soporte y disponibilidad de software de código abierto sin coste alguno.

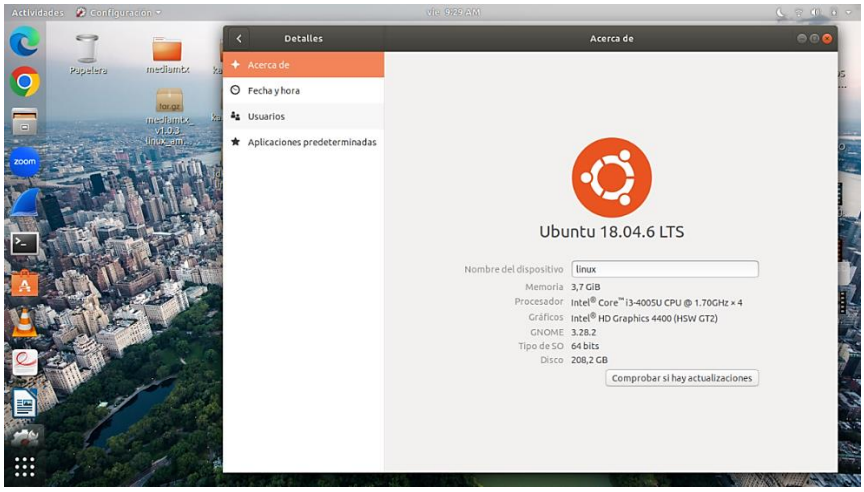


Figura 12. *Interfaz de Ubuntu 18.04.6 LTS SO.*
Fuente: *Elaborado por el Autor.*

3.1.1.2 PYTHON

Python es un lenguaje de programación de alto nivel, el cual es versátil y de propósito general. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python es conocido por su legibilidad y simplicidad, lo que lo hace popular entre los principiantes en programación debido a que es amigable y fácil de aprender. Además, es ampliamente utilizado en campos como la ciencia de datos, el aprendizaje automático y el desarrollo web. Python tiene una gran comunidad de desarrolladores y una amplia variedad de bibliotecas y marcos disponibles para facilitar el desarrollo de aplicaciones. [59]

3.1.1.3 CONTROLADORES SDN

3.1.1.4 OPENDAYLIGHT

OpenDaylight (ODL) es una plataforma modular abierta para personalizar y automatizar redes de cualquier tamaño y escala. El proyecto OpenDaylight surgió del movimiento SDN, con un claro enfoque en la programabilidad de la red. Fue diseñado desde el principio como base para soluciones comerciales que abordan una variedad de casos de uso en entornos de red existentes. ODL incluye soporte para el conjunto más amplio de protocolos en cualquier plataforma SDN (OpenFlow, OVSDB, NETCONF, BGP y muchos más) que mejoran la programabilidad de las redes modernas y resuelven una variedad de necesidades de los usuarios. OpenDaylight cuenta con 13 lanzamientos, más de 1000 autores/remitentes y potencia redes de más de mil millones de suscriptores globales. Como parte de LF Networking, ODL está impulsado por una comunidad colaborativa global de organizaciones de proveedores y usuarios que se adapta continuamente para soportar el conjunto más amplio de casos de uso de SDN y NFV de la industria. [60]

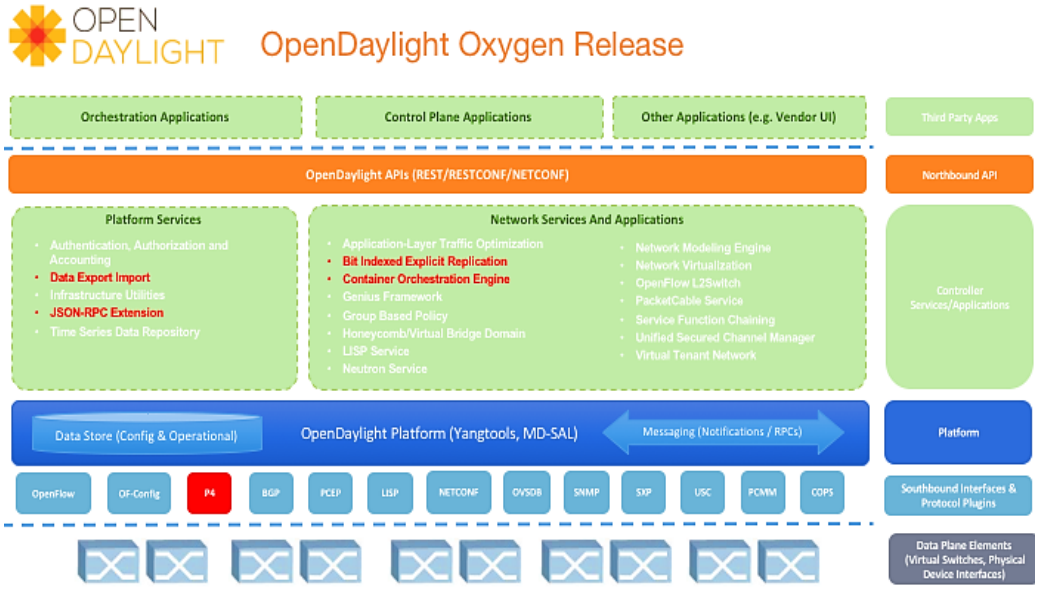


Figura 13. *OpenDaylight.*

Fuente: *Extraído del sitio web de OpenDayLight.*

DLUX

DLUX proporciona una interfaz gráfica de usuario intuitiva para el controlador de red OpenDaylight. Esta interfaz de usuario, que no mantiene estado, se comunica con el servicio backend para proporcionar una interfaz coherente y fácil de usar.

Esta interfaz permite interactuar de manera eficiente con los proyectos y el controlador base de OpenDaylight. [61]

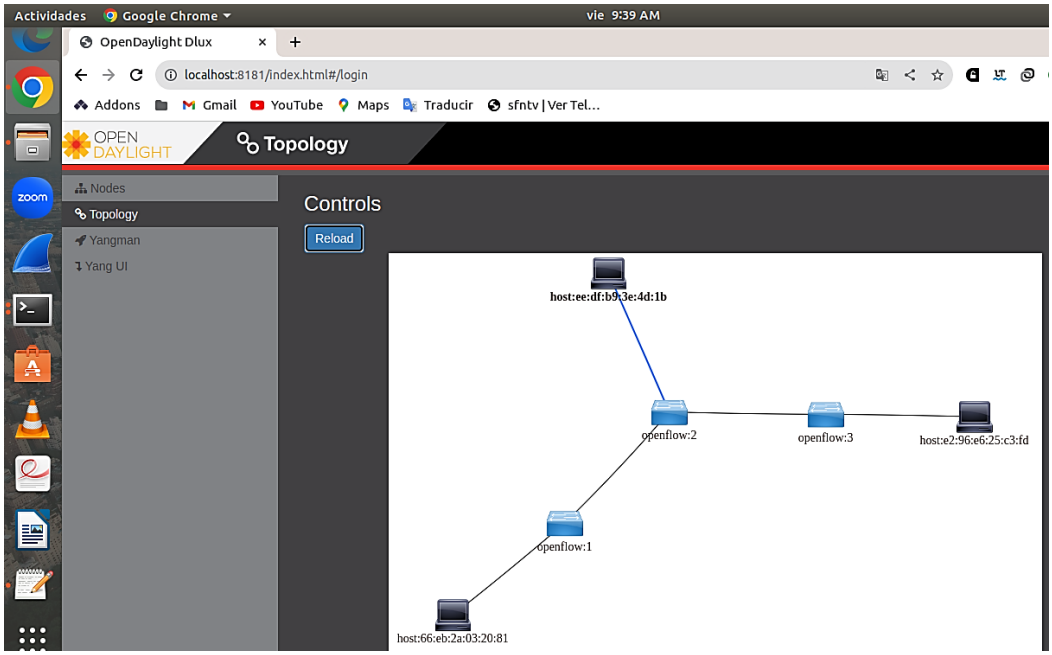


Figura 14. *Interfaz DLUX de ODL, vista de topología de red.*
Fuente: *Elaborado por el Autor.*

JAVA

Para poner en funcionamiento el servicio de ODL en la versión karaf.0.8.4, es imprescindible instalar la versión 8 de Java (JDK 8u202). Esto se debe a que ODL fue desarrollado utilizando el lenguaje de programación Java. Se requiere del entorno de ejecución de Java (Java Runtime Environment, JRE) que proporciona los requisitos mínimos para su ejecución. El Kit de Desarrollo de Java (Java Development Kit, JDK) es un entorno destinado al desarrollo de aplicaciones que utilizan el lenguaje de programación Java. Además, el JDK incluye el JRE, así como herramientas para depurar y monitorear aplicaciones Java. También proporciona una amplia documentación y ejemplos de código para asistir a los desarrolladores en la creación de aplicaciones robustas y eficientes. [62]

3.1.1.5 POX

POX es un controlador de red de código abierto diseñado para redes definidas por software (SDN). Esta herramienta permite a los desarrolladores diseñar aplicaciones de red personalizadas y controlar el comportamiento de la red programando dispositivos de red compatibles con OpenFlow. POX es una plataforma ligera y fácil de usar que ofrece una API sencilla para el desarrollo de

aplicaciones SDN. Está desarrollado en Python y, aunque inicialmente se diseñó como un controlador SDN OpenFlow, ahora también puede funcionar como un switch OpenFlow y puede ser útil para la creación de software de red en general. Se utilizó la versión 0.3.0 (dart) de POX, desarrollada por James McCauley, que requiere Python 2.7.17. Un aspecto destacado del diseño de POX es su facilidad de instalación y ejecución. [63]

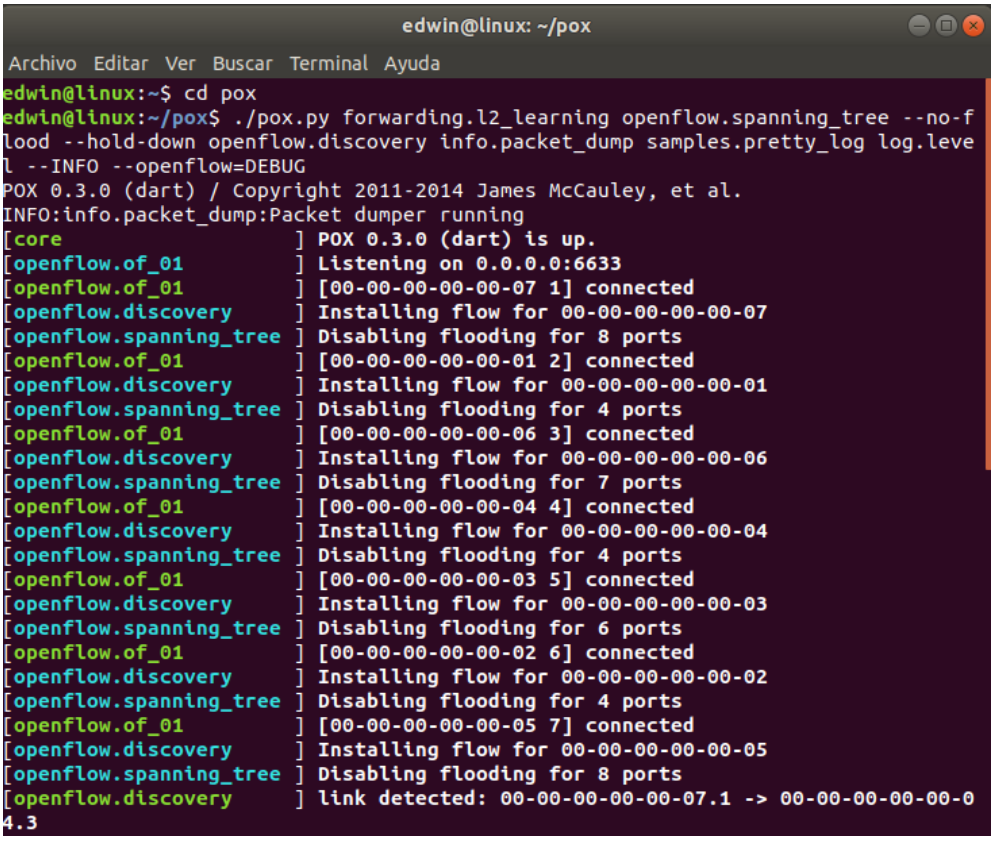


Figura 15. *Interfaz CLI del controlador POX.*
Fuente: *Elaborado por el Autor.*

MÓDULO GROUPFLOW

El módulo GroupFlow proporciona enrutamiento de multidifusión en redes definidas por software. Este proyecto tiene como objetivo la creación de una bifurcación POX de la implementación NOX-Classic Castflow, complementada con soporte completo para IGMPv3 implementado en el controlador. Esto permite la implementación de propuestas de multidifusión SDN en una red OpenFlow SDN en vivo con hosts IGMPv3 estándar. El objetivo final de este proyecto es ofrecer un banco de pruebas para los desarrolladores que deseen experimentar con nuevas ideas de multidifusión en una red definida por software. GroupFlow se

implementa como un conjunto de módulos para POX en un controlador SDN basado en Python. [64]

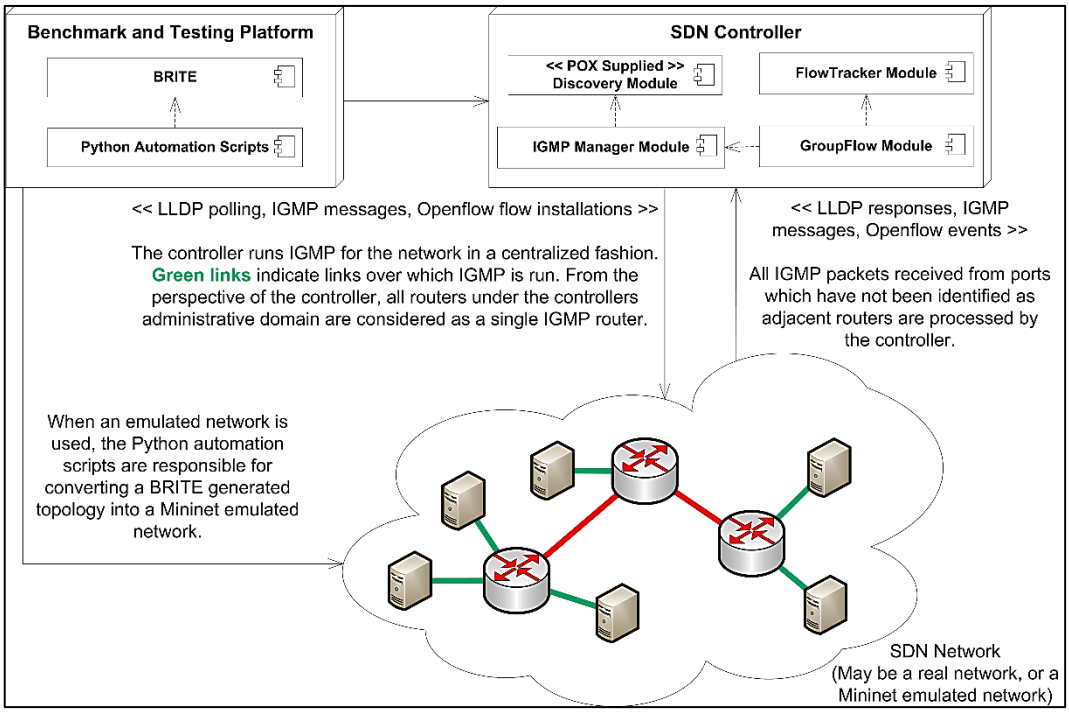


Figura 16. *Conjunto de módulos integrados al controlador SDN POX.*
Fuente: *Extraído del repositorio de GitHub GroupFlow.*

3.1.1.6 MININET

Mininet es un software emulador de red que facilita la creación y simulación de redes virtuales de computadoras. Con Mininet, es posible simular una red completa con todos sus componentes, como hosts, switches, enlaces y controladores, utilizando únicamente un PC o una máquina virtual (VM). Mininet es una herramienta valiosa para realizar pruebas y experimentos con diferentes configuraciones y protocolos de red, así como para aprender sobre la lógica y el funcionamiento de las redes. Mininet utiliza el concepto de contenedores de Linux, que son espacios aislados que utilizan el mismo núcleo del sistema operativo. Cada contenedor funciona como un nodo de la red, y se le pueden asignar recursos como CPU, memoria y ancho de banda. Mininet también ofrece la posibilidad de conectar la red virtual con la red real, lo que permite integrar otras herramientas y aplicaciones. Además, Mininet es una excelente forma de desarrollar, compartir y experimentar con sistemas de redes definidas por software (SDN) utilizando OpenFlow. [65]

Para la realización de esta investigación, se empleó la versión 2.3.1b1 de MiniNet.

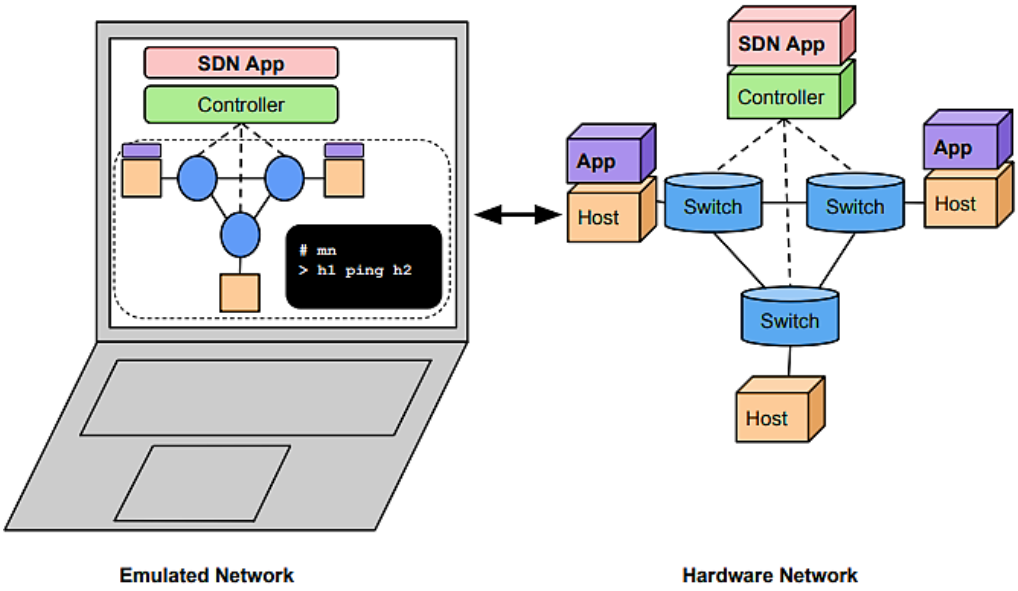


Figura 17. *MiniNet. [66]*
Fuente: *Extraído del sitio web de Open Networking Foundation.*

3.1.1.7 OPEN VSWITCH

Open vSwitch es un conmutador virtual de software de código abierto que se utiliza para proporcionar conectividad de red en entornos virtualizados. Este conmutador está diseñado para permitir la automatización de la red a gran escala a través de programación y soporta múltiples protocolos de control, incluyendo OpenFlow. En el contexto de Mininet, Open vSwitch está integrado y se utiliza como una opción para el intercambio de paquetes a través de interfaces. Esto posibilita a los usuarios la creación de topologías de red más complejas y la experimentación con diferentes protocolos de control. [67]

3.1.1.8 MINIEDIT:

Miniedit, una herramienta gráfica perteneciente a Mininet, proporciona una plataforma sencilla e intuitiva para la creación y ejecución de simulaciones de Redes Definidas por Software (SDN). Esta herramienta permite el diseño de topologías de red mediante un proceso de arrastrar y soltar elementos como hosts, switches, routers y controladores en un lienzo. Cada uno de estos elementos puede ser configurado con propiedades específicas, incluyendo la dirección IP, el nombre, el protocolo OpenFlow y el controlador asociado. Una vez que la topología ha sido diseñada, Miniedit ofrece la posibilidad de iniciar la simulación y evaluar el funcionamiento de la red mediante comandos o aplicaciones.

Adicionalmente, Miniedit permite guardar y cargar topologías en formato Python, lo que facilita su modificación y reutilización posterior. [68]

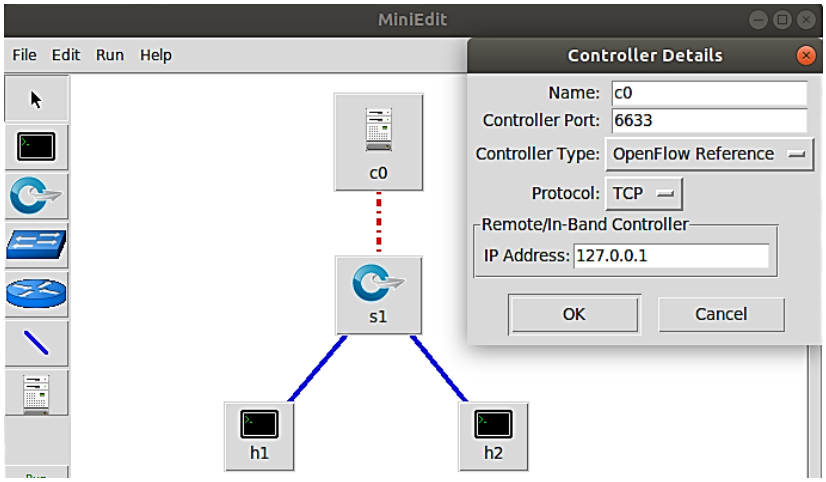


Figura 18. *Interfaz gráfica de Miniedit.*
Fuente: *Elaborado por el Autor.*

3.1.1.9 VLC MEDIA PLAYER

VLC es un reproductor multimedia de código abierto, gratuito y multiplataforma, además de un marco de trabajo que permite la reproducción de la mayoría de los archivos multimedia, incluyendo DVD, Audio CD, VCD y diversos protocolos de transmisión para vídeos en directo en la red, ya sea en unidifusión o multidifusión. Este reproductor ha sido desarrollado por voluntarios de la comunidad VideoLAN. VLC utiliza sus propios códecs internos y es compatible con todas las plataformas populares. Puede leer casi todos los archivos multimedia, transmisiones de red, tarjetas capturadoras y otros formatos de archivos. VLC puede funcionar tanto como servidor y como cliente para transmitir y recibir transmisiones de red, pudiendo transmitir todo lo que puede leer. [69]

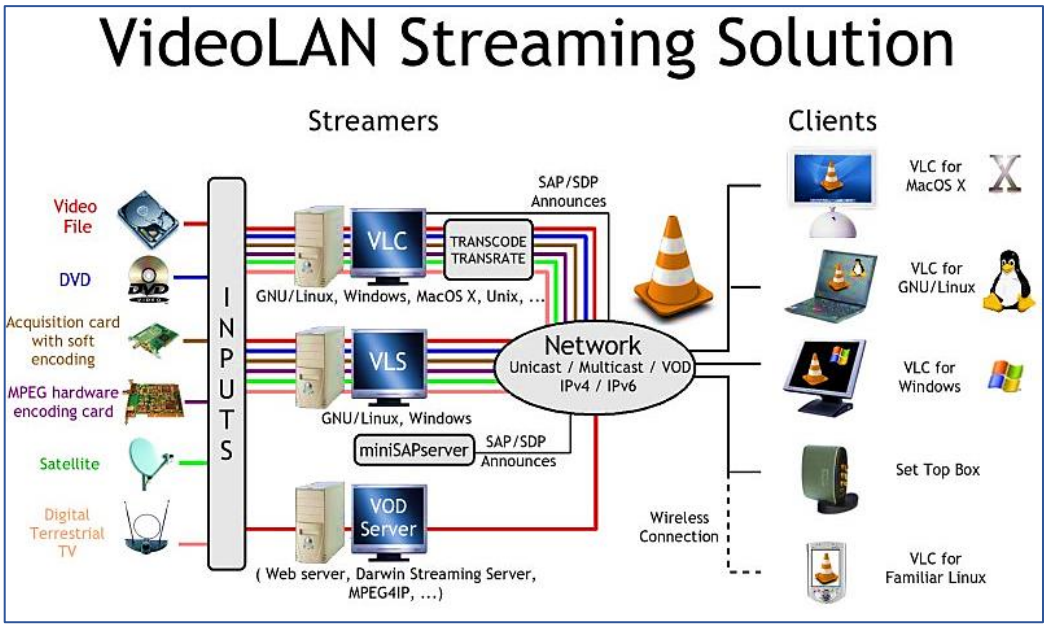


Figura 19. *Solución para streaming de VideoLAN.*
Fuente: *Extraído del sitio web de VideoLAN.*

A continuación, se presentan ejemplos del ancho de banda (BW) necesario para la transmisión de video en streaming utilizando el proyecto de VideoLAN:

Tabla 9. *Ancho de banda necesario para streaming en VideoLAN.*
Fuente: *Extraído del sitio web de VideoLAN.*

ANCHO DE BANDA (BW)	FORMATO DE TRANSMISIÓN
0,5 - 4 Mbps	Transmisión en MPEG-4.
3 - 4 Mbps	Para un flujo MPEG-2 leído desde una tarjeta de satélite, una tarjeta de televisión digital o una tarjeta de codificación MPEG-2
6 - 9 Mbps	Para un DVD.

3.1.1.10 FFMPEG

FFmpeg es un marco multimedia líder, con la capacidad de decodificar, codificar, transcodificar, multiplexar, demultiplexar, transmitir, filtrar y reproducir prácticamente cualquier cosa que los humanos y las máquinas hayan creado. Es compatible con una amplia gama de formatos, desde los más antiguos y oscuros hasta los más modernos, independientemente de si fueron diseñados por un comité de estándares, la comunidad o una corporación. FFmpeg es altamente portátil: se compila, se ejecuta y pasa la infraestructura de prueba FATE en Linux, Mac OS, Microsoft Windows, BSD, Solaris, etc., en una amplia variedad de entornos de

compilación, arquitecturas de máquinas y configuraciones. El proyecto FFmpeg se esfuerza por proporcionar la mejor solución técnica posible tanto para los desarrolladores de aplicaciones como para los usuarios finales. Para lograr este objetivo, combina las mejores opciones de software gratuito disponibles. [70]

3.1.1.11 MEDIAMTX

MediaMTX es un servidor de medios y un proxy de medios en tiempo real, independiente y listo para usar, que permite la publicación, lectura, representación y grabación de transmisiones de video y audio. Ha sido diseñado como un "enrutador de medios", que dirige flujos de medios de un extremo a otro. Las transmisiones en vivo pueden ser publicadas en el servidor utilizando diversos protocolos, incluyendo RTMP, RTSP y UDP. [71]

3.1.1.12 WIRESHARK:

Wireshark es un programa de análisis de paquetes de red, que permite visualizar en detalle el contenido de paquetes capturados que circulan por una red de computadoras con el mayor detalle posible. Se podría pensar en un analizador de paquetes de red como un dispositivo de medición para examinar lo que sucede dentro de un cable de red, tal como un electricista usa un voltímetro para examinar lo que sucede dentro de un cable eléctrico (pero a un nivel superior, por supuesto). En el pasado, estas herramientas eran muy caras, patentadas o ambas cosas. Sin embargo, con la llegada de Wireshark eso ha cambiado y se ha democratizado el acceso a esta tecnología. Esta herramienta, disponible de forma gratuita y de código abierto es uno de los mejores analizadores de paquetes de red disponibles en la actualidad. [72]

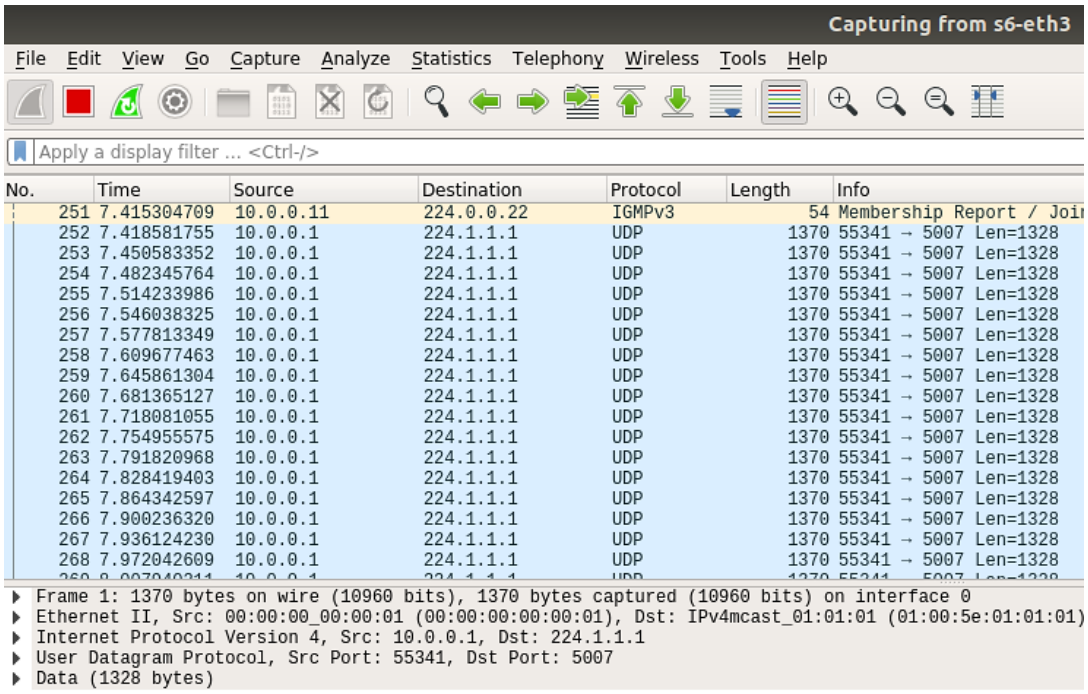


Figura 20. *Interfaz de captura de paquetes en Wireshark.*
Fuente: *Elaborado por el Autor.*

3.1.1.13 IPERF

iPerf es una herramienta para mediciones activas del ancho de banda máximo alcanzable en redes IP. Admite el ajuste de varios parámetros relacionados con la sincronización, los buffers y los protocolos (TCP, UDP, SCTP con IPv4 e IPv6). Para cada prueba, informa el ancho de banda, la pérdida y otros parámetros. iPerf fue desarrollado originalmente por NLANR/DAST. iPerf está desarrollado principalmente por ESnet / Laboratorio Nacional Lawrence Berkeley. [73]

3.2 DISEÑO DE LA PROPUESTA

Para el desarrollo de escenarios de simulación en un entorno de virtualización de red destinado a un servicio de transmisión de video en streaming, es crucial conceptualizar el diseño de las topologías de la red y ejecutar las emulaciones correspondientes en MiniNet, utilizando una plataforma de controlador de red SDN (OpenDayLight - POX). Este proceso debe incluir la integración de los protocolos y recursos esenciales para la implementación exitosa del servicio de streaming multicast. Este enfoque garantiza una transmisión eficiente y de alta calidad.

3.2.1 DISEÑO DE ESCENARIOS SDN IMPLEMENTANDO LOS CONTROLADORES DE RED ODL Y POX EN DIFERENTES TOPOLOGÍAS DE RED:

Con el propósito de desarrollar los escenarios de simulación, se ha diseñado tres topologías de red SDN de tipo lineal, árbol y malla, las cuales están compuestas por los siguientes elementos: un controlador de red SDN para cada topología de red (ODL - POX), tres servidores de streaming (serv1, serv2 y serv3), nueve estaciones que desempeñan el papel de clientes finales (h1, h2, h3, h4, h5, h6, h7, h8 y h9), tres switches habilitados con OpenFlow para una topología lineal y siete switches para una topología árbol y malla. En las figuras 21, 22 y 23 se muestran gráficos donde se detalla cada escenario de topología de red desarrollado que abarca las interfaces de cada dispositivo de red, el ancho de banda asignado a cada enlace, las direcciones IP y nombres asignados a cada nodo.

Los escenarios de red SDN se diseñaron y evaluaron bajo dos condiciones distintas: la primera, manteniendo los parámetros de los enlaces de la topología de red en sus valores predeterminados; la segunda, introduciendo variaciones en los enlaces de la topología de red mediante la adición de valores de retardo y pérdida de paquetes.

3.2.1.1 ESQUEMA DE ESCENARIO DE TOPOLOGÍA DE RED SDN LINEAL:

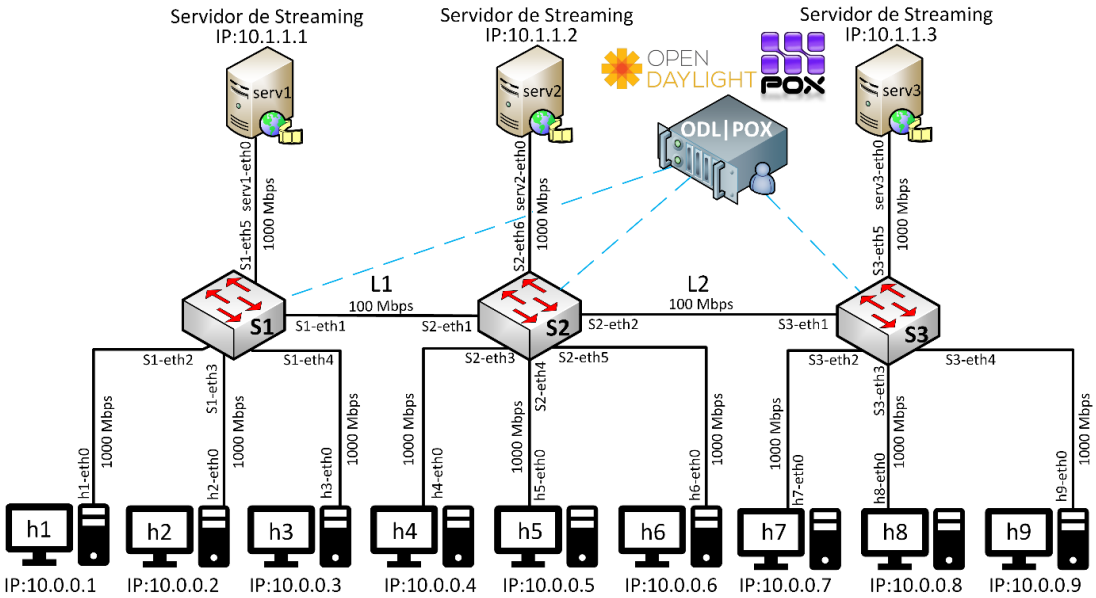


Figura 21. *Escenario SDN con topología de red lineal.*
Fuente: *Elaborado por el Autor.*

3.2.1.2 ESQUEMA DE ESCENARIO DE TOPOLOGÍA DE RED SDN ARBOL:

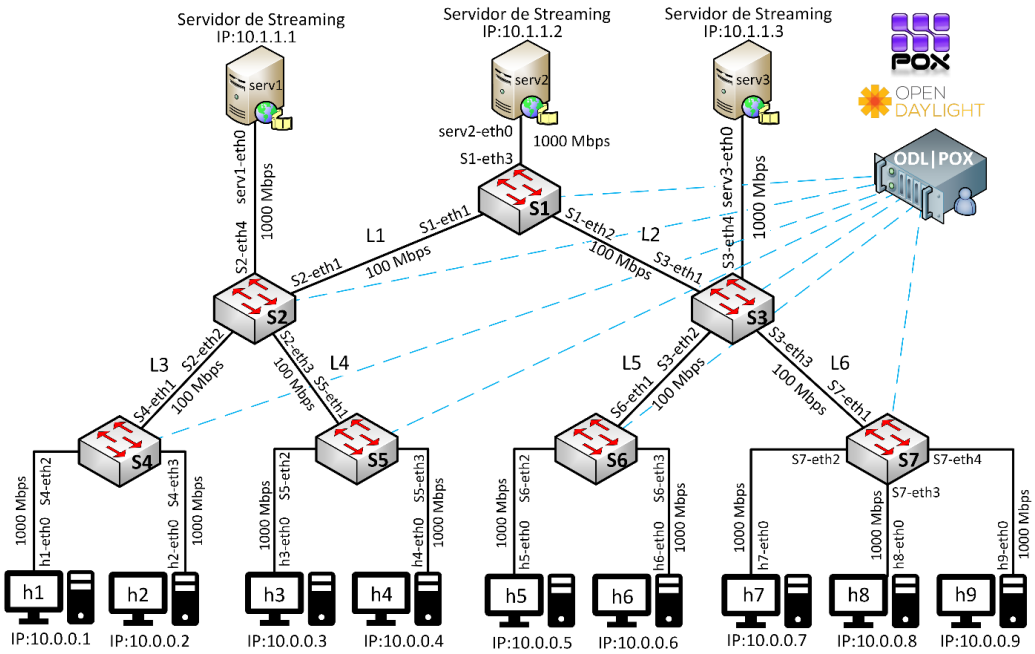


Figura 22. *Escenario SDN con topología de red árbol.*
Fuente: *Elaborado por el Autor.*

3.2.1.3 ESQUEMA DE ESCENARIO DE TOPOLOGÍA DE RED SDN MALLA:

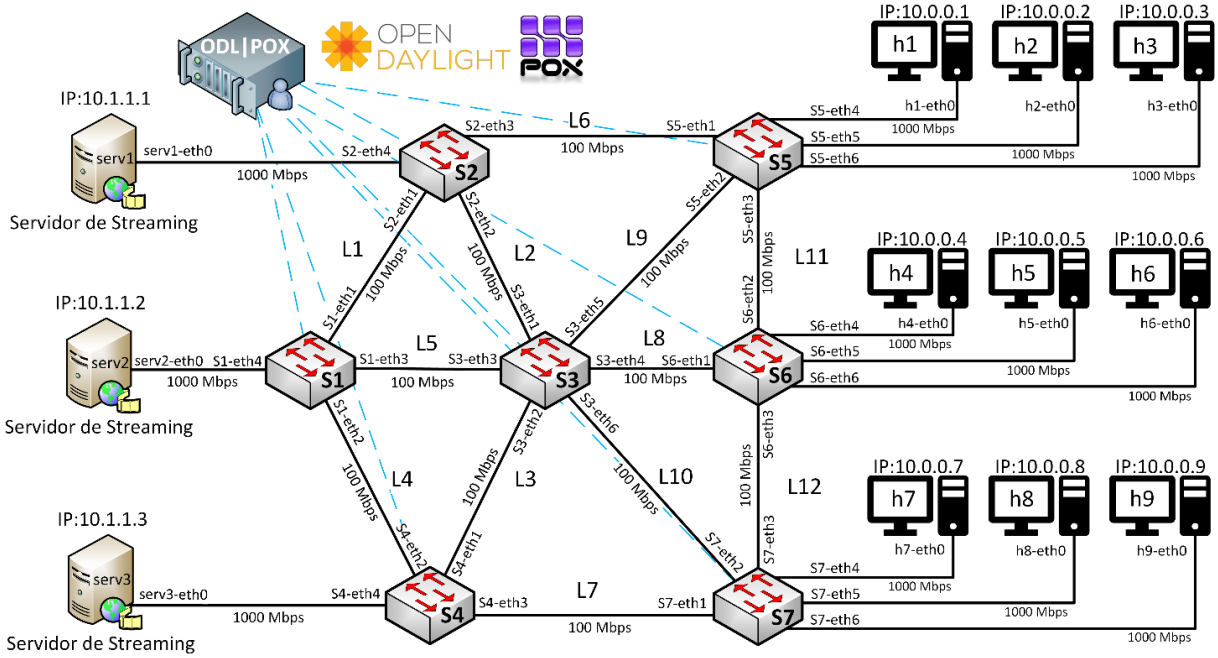


Figura 23. *Escenario SDN con topología de red malla.*
Fuente: *Elaborado por el Autor.*

3.2.2 ESCENARIOS SDN CON VARIACIÓN EN PARAMETROS DE LA TOPOLOGÍA DE RED

Para cada escenario de red, se introdujeron variaciones en los parámetros de cada enlace de las distintas topologías de red. Estos parámetros incluyen el valor de retardo (delay), expresado en milisegundos (ms), y el valor de pérdida de paquetes (Packet Loss), expresado en porcentaje (%). Los valores aplicados a cada escenario de red desarrollado se detallan en la Tabla 10.

Tabla 10. *Parámetros de escenarios de topología de red SDN.*
Fuente: *Elaborado por el Autor.*

Topología de red	Enlace	BW	Tasa de pérdida	Retardo
Lineal	L1	100 Mbps	5%	10ms
	L2	100 Mbps	10%	5ms
Árbol	L1	100 Mbps	1%	1ms
	L2	100 Mbps	1%	1ms
	L3	100 Mbps	10%	5ms
	L4	100 Mbps	5%	10ms
	L5	100 Mbps	10%	5ms
	L6	100 Mbps	5%	10ms
Malla	L1	100 Mbps	10%	1ms
	L2	100 Mbps	5%	3ms
	L3	100 Mbps	3%	5ms
	L4	100 Mbps	1%	10ms
	L5	100 Mbps	10%	1ms
	L6	100 Mbps	5%	3ms
	L7	100 Mbps	3%	5ms
	L8	100 Mbps	1%	10ms
	L9	100 Mbps	10%	1ms
	L10	100 Mbps	5%	3ms
	L11	100 Mbps	3%	5ms
	L12	100 Mbps	1%	10ms

3.2.3 CONFIGURACIÓN DE CONTROLADORES DE RED SDN

3.2.3.1 CONTROLADOR DE RED OPENDAYLIGHT:

Para el desarrollo de los escenarios de simulación, se empleó la versión Oxygen karaf-0.8.4 del controlador de red OpenDaylight. Para llevar a cabo la configuración, iniciamos un nuevo terminal en nuestro entorno Ubuntu 18.04.6 LTS y ejecutamos el comando correspondiente.

```
> sudo su
```

‘Sudo su’ es una utilidad en el sistema operativo Linux que permite realizar acciones con los privilegios y permisos del usuario ‘root’ (también conocido como superusuario). Cuando se utiliza el comando ‘su’ sin especificar un nombre de usuario, se asume por defecto que se desea acceder como superusuario. En este caso, nos dirigimos a la ubicación de la carpeta ‘karaf-0.8.4’, que contiene el software del controlador ODL. Esta carpeta se encuentra en el directorio denominado ‘Escritorio’. A continuación, ejecutamos el software del controlador ODL utilizando los comandos correspondientes.

```
> cd Escritorio/karaf-0.8.4
> ./bin/karaf
```

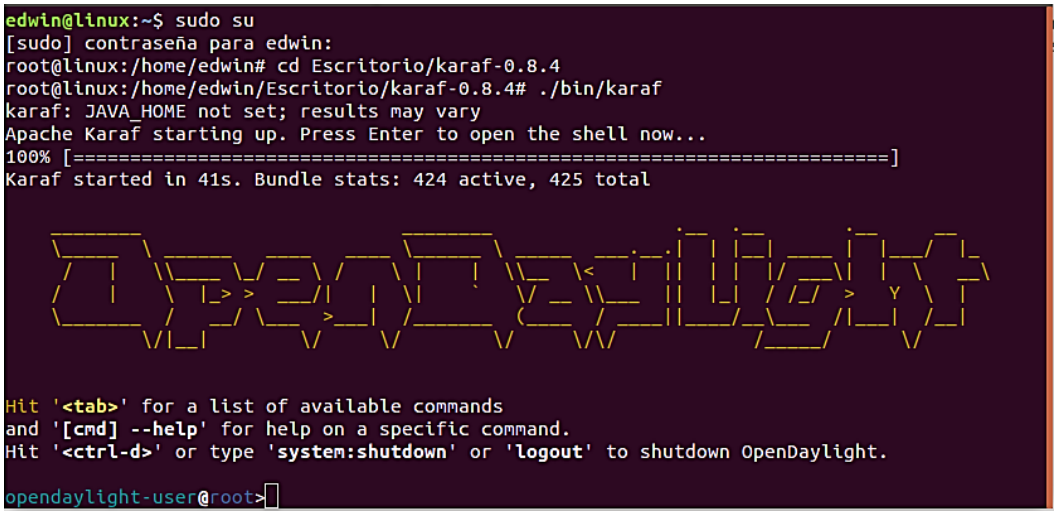


Figura 24. *Karaf; Interfaz de consola de ODL.*
Fuente: *Elaborado por el Autor.*

OpenDaylight se inicia de manera predeterminada sin ninguna función. Para enumerar los tipos de funciones disponibles, las distribuciones de lanzamiento de ODL ofrecen la posibilidad de seleccionar entre todas las funciones para su instalación. Para ello, se ejecuta el comando correspondiente en la consola Karaf de ODL.

```
opendaylight-user@root> feature:list
```

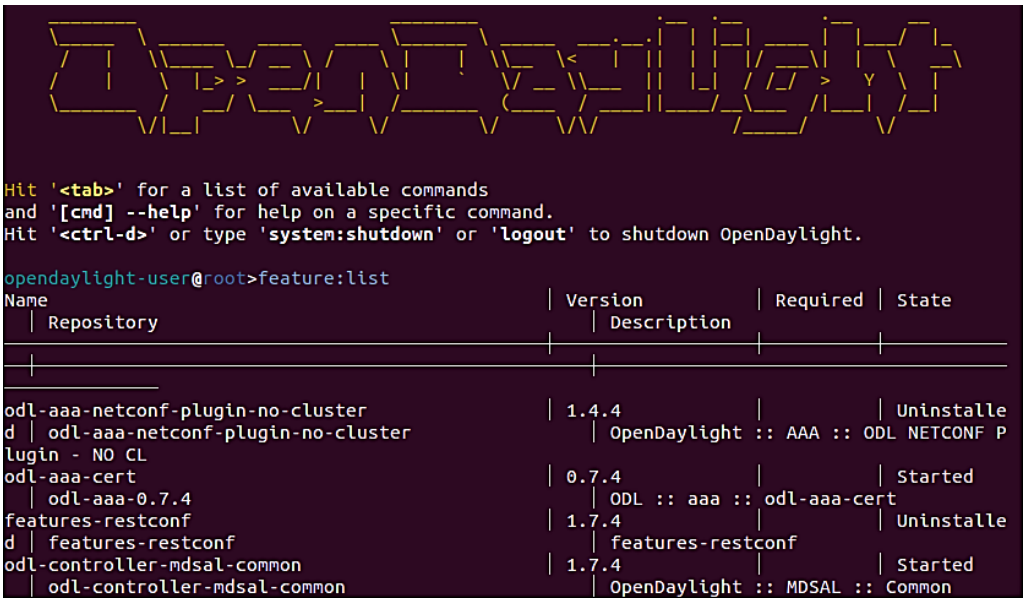


Figura 25. *Funciones de ODL disponibles.*
Fuente: *Elaborado por el Autor.*

Se han seleccionado las funciones esenciales de OpenDaylight para el desarrollo específico de los escenarios de simulación. Estas funciones se detallan en la tabla siguiente:

Tabla 11. *Complementos del controlador OpenDaylight.*
Fuente: *Información extraída del sitio web de OpenDaylight.*

FUNCIÓN DE KARAF (ODL)	DESCRIPCIÓN
odl-dluxapps-nodes	La aplicación muestra una lista de nodos en la topología OpenFlow.
odl-dluxapps-topology	Aplicación de topología básica. Muestra nodos y enlaces desde la topología OpenFlow.
odl-dlux-core	Activa la interfaz de usuario de DLUX. Instala internamente la aplicación de topología DLUX, junto con los componentes principales.
odl-l2switch-switch-ui	Esta función ejecuta L2 switch dentro de la distribución OpenDaylight.
odl-l2switch-all	L2 switch proporciona funcionalidad de conmutador capa 2: Controlador de paquetes, removedor de bucles, rastreador de direcciones IP, instalador de flujos según el tráfico de red.

[74]

Para agregar las funciones necesarias, se ejecutan los siguientes comandos en la consola Karaf de OpenDaylight:

```
opendaylight-user@root> feature:install odl-l2switch-switch-ui odl-l2switch-all odl-dlux-core odl-dluxapps-nodes odl-dluxapps-topology
```

Para acceder a la interfaz gráfica de OpenDaylight (ODL), es imprescindible iniciar sesión en DLUX. Para ello, se debe abrir un navegador web y dirigirse al URL de inicio de sesión, que es "http://localhost:8181/index.html". Aunque se puede utilizar cualquier navegador, se recomienda el uso de Google Chrome para una experiencia óptima. Una vez en la página de inicio de sesión, se deben introducir las credenciales correspondientes en los campos de nombre de usuario y contraseña. Las credenciales predeterminadas para OpenDaylight son 'admin' tanto para el nombre de usuario como para la contraseña.

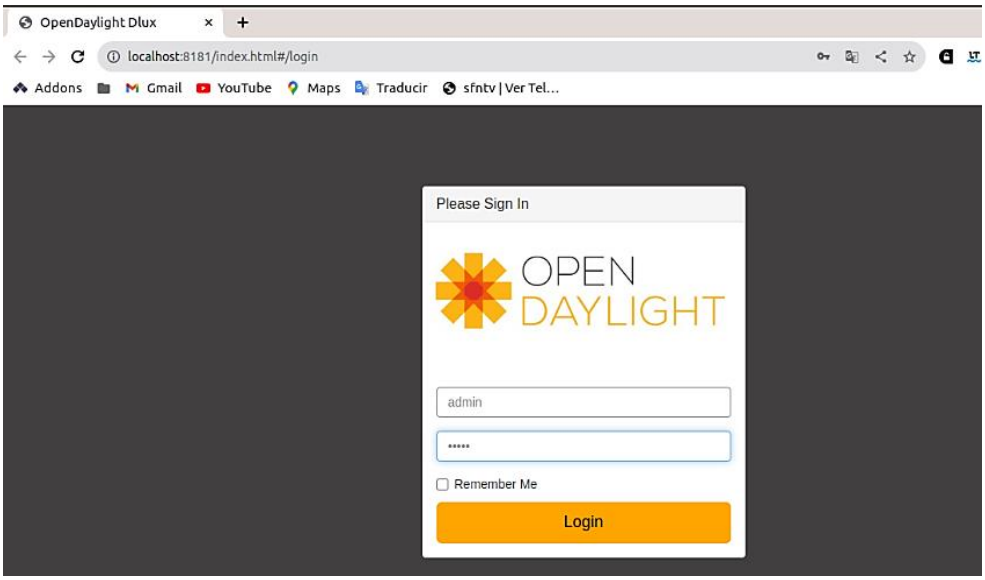


Figura 26. *Ventana de inicio de sesión de ODL.*
Fuente: *Elaborado por el Autor.*

Una vez que se ha iniciado sesión en DLUX, los módulos disponibles se mostrarán en el panel situado en el lado izquierdo de la interfaz.

Visualización de estadísticas de red

El módulo Nodos del panel izquierdo nos permite ver las estadísticas de red y la información de puertos de los conmutadores de la red.

Para utilizar el módulo nodos seleccionamos nodos en el panel izquierdo. El panel derecho muestra una tabla que enumera todos los nodos, conectores de nodo y las estadísticas. Introducimos un ID de nodo en la ficha "buscar nodos" para buscar por conectores de nodo. Hacemos clic en el número de conector de nodo para ver detalles como el ID del puerto, el nombre del puerto, el número de puertos por

switch, la dirección MAC, etc. Hacemos clic en “flows” en la columna estadísticas para ver las estadísticas de la tabla de flujo para el nodo en particular: como ID de tabla, coincidencia de paquetes, flujos activos, etc. Hacemos clic en “conectores de nodo” para ver las estadísticas del conector de nodo para el ID de nodo en concreto.

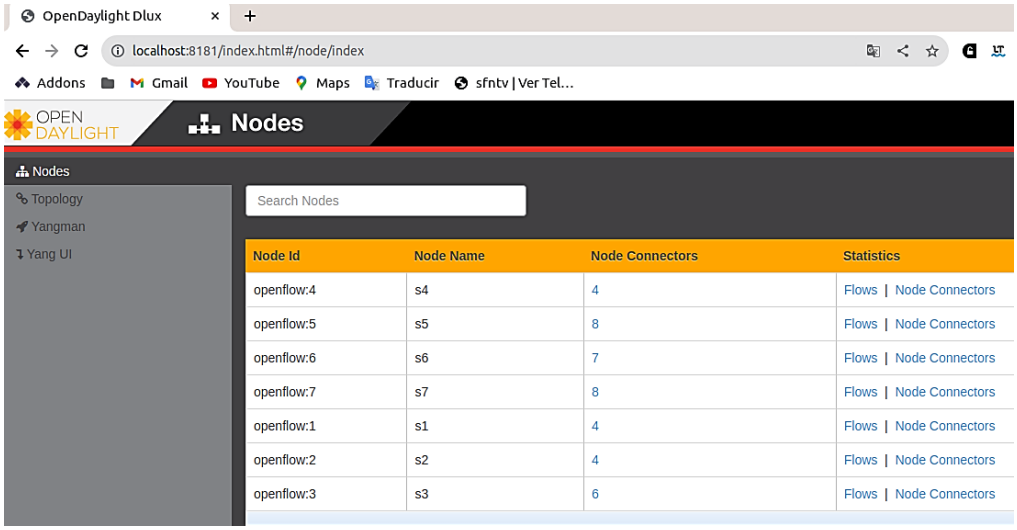


Figura 27. *Módulo de nodos en la interfaz de ODL.*
Fuente: *Elaborado por el Autor.*

Visualización de la topología de red

El módulo de topología muestra una representación gráfica de la topología de red creada. DLUX no permite editar o agregar información de topología. La topología de red se genera y edita sobre MiniNet y DLUX puede leerla y mostrarla.

Para ver la topología de red seleccionamos el módulo de topología en el panel izquierdo. Veremos la representación gráfica en el panel derecho. En el diagrama, los cuadros azules representan los switches, el negro representa los hosts disponibles y las líneas representan los enlaces de conexión entre switches y los hosts. Pasamos el ratón sobre los hosts, enlaces (links) o switches para ver los puertos de origen y destino.

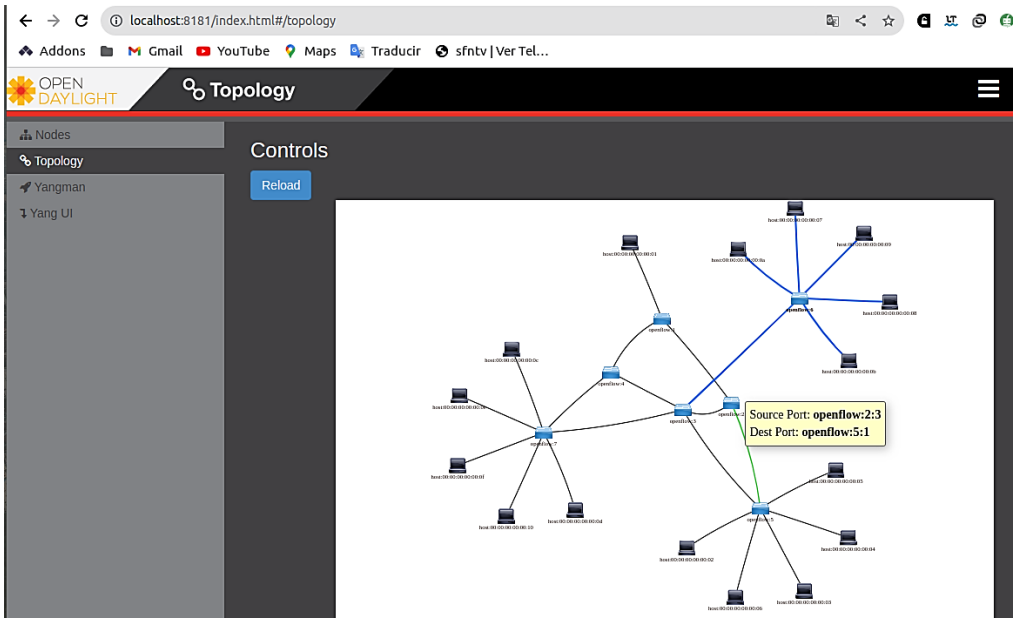


Figura 28. *Módulo de topología de red en la interfaz de ODL.*
Fuente: *Elaborado por el Autor.*

3.2.3.2 CONTROLADOR DE RED POX:

El controlador de red POX por sí solo no dispone de muchas funcionalidades, sin embargo dispone de componentes que se pueden emplear para diferentes propósitos.

En lo que se refiere a componentes en POX, lo que realmente se quiere decir es algo que podemos poner y ejecutar en la interfaz de línea de comandos (CLI) del controlador de red. De manera predeterminada viene con una serie de componentes de stock. Algunos de estos proporcionan funcionalidad básica y algunos proporcionan características convenientes. La siguiente tabla muestra una lista de componentes utilizados en el desarrollo de esta investigación.

Tabla 12. *Complementos de POX.*
Fuente: *Extraído de repositorio GitHub, POX Manual Current documentation.*

COMPONENTE	DESCRIPCIÓN DE CARACTERÍSTICA
forwarding.l2_learning	Este componente hace que los switches OpenFlow actúen como un tipo de switch de aprendizaje de capa 2 (L2). Si bien este componente aprende direcciones L2, los flujos que instala coinciden exactamente en tantos campos como sea posible.
openflow.spanning_tree	Este componente utiliza el componente de detección para crear una vista de la topología de red, construye un árbol de expansión y a continuación deshabilita la inundación en los

	puertos del conmutador que no están en el árbol. El resultado es que las topologías con bucles ya no son un problema. Este componente tiene dos opciones que alteran el comportamiento de inicio: **--no-flood:** desactiva la inundación en todos los puertos tan pronto como se conecta un switch; En algunos puertos, se habilitará más adelante. **--hold-down:** evita la alteración del control de inundaciones hasta que se haya completado un ciclo de descubrimiento completo (por lo tanto, todos los enlaces tienen la oportunidad de ser descubiertos).
openflow.discovery	Este componente envía mensajes LLDP especialmente diseñados desde los switches OpenFlow para que pueda descubrir la topología de red. Genera eventos (que puedes escuchar) cuando los enlaces están habilitados o caídos. Más específicamente, puede escuchar eventos cuando se detecta un enlace (link).
info.packet_dump	Un componente simple que muestra paquetes de entrada como información en el registro. Algo así como ejecutar tcpdump en un switch.
samples.pretty_log	Este sencillo módulo utiliza log.color el cual colorea el registro y emplea un formato de registro personalizado para proporcionar una salida de registro agradable y funcional en la consola.
log.level	POX utiliza la infraestructura de registro de Python. El componente log.level permite configurar el nivel de detalle que muestran determinados registradores. Los niveles de mayor a menor gravedad son: **CRITICAL** **ERROR** **WARNING** **INFO** **DEBUG**

[75]

GroupFlow: Enrutamiento de multidifusión en redes definidas por software (SDN).

Para la implementación de un servicio de transmisión en streaming de tipo multicast en la plataforma MiniNet, es esencial que el controlador de red POX esté equipado con el módulo GroupFlow. Este módulo debe ser inicializado en conjunto con los módulos "openflow.discovery", "openflow.igmp_manager", "openflow.flow_tracker" y "openflow.groupflow". A continuación, se presenta una tabla que describe los módulos del proyecto GroupFlow que se han utilizado en el desarrollo de esta investigación.

Tabla 13. *Módulos del proyecto GroupFlow.*
Fuente: *Extraído del repositorio GitHub del proyecto GroupFlow.*

COMPONENTE	DESCRIPCIÓN DE CARACTERÍSTICA
openflow.flow_tracker	Un módulo POX que consulta periódicamente la red para estimar la utilización del enlace. Depende de openflow.discovery. Se admiten los siguientes argumentos de línea de comando: **query_interval:** el período de tiempo (en segundos) que debe transcurrir entre consultas consecutivas de estadísticas de flujo/puerto a los conmutadores. Predeterminado: 2 **link_max_bw:** el ancho de banda máximo (en Mbps) de los enlaces de red. Tenga en cuenta que en la implementación actual, todos los enlaces deben tener un ancho de banda uniforme. Predeterminado: 30 **link_cong_threshold:** la utilización (en Mbps) por encima de la cual un enlace debe considerarse congestionado. Debe ser <= link_max_bw. Predeterminado: 28,5 **avg_smooth_factor:** el valor Alpha que se utilizará para el promedio exponencial de estimación del ancho de banda. Predeterminado: 0,7 **log_peak_usage:** (Verdadero/Falso) Si es verdadero, la utilización máxima y promedio en la red se registran en debug.info en un intervalo de (query_interval / 1,5) segundos. Valor predeterminado: Falso.

openflow.igmp_manager	Una implementación del módulo POX que proporciona la funcionalidad de gestión de grupos multicast con IGMPv3. Este módulo no admite ningún argumento de línea de comando. Todos los parámetros IGMP están configurados según los valores predeterminados recomendados por RFC (consulte RFC 3376). Depende de openflow.discovery
openflow.groupflow	Una implementación del módulo POX de enrutamiento multicast, respaldada por la gestión del estado del grupo mediante el módulo administrador de IGMP. El módulo GroupFlow calculará un árbol de ruta más corta utilizando el algoritmo Dijkstra desde la fuente de multidifusión a todos los enrutadores de la red. **util_link_weight:** determina el factor de escala por el cual se multiplicará el peso del enlace basado en la utilización. Tenga en cuenta que establecer esto en 0 con cualquier tipo de peso de enlace producirá únicamente árboles de menor costo en términos de número de saltos. Predeterminado: 10 **link_weight_type:** determina el método mediante el cual se escalan los pesos de los enlaces con la utilización del enlace. Las opciones admitidas son "lineal" (el peso del enlace aumenta como una función lineal) o "exponencial" (el peso del enlace crece exponencialmente al aumentar la utilización). Predeterminado: lineal

[76]

Una vez que se han establecido los componentes necesarios, accedemos al directorio POX y ejecutamos el controlador de red en un terminal con los parámetros correspondientes.

```
> ./pox.py forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down openflow.discovery info.packet_dump samples.pretty_log openflow.flow_tracker --query_interval=1 --link_max_bw=30 --link_cong_threshold=30 --avg_smooth_factor=0.65 --log_peak_usage=False openflow.igmp_manager openflow.groupflow --util_link_weight=10 –link_weight_type=linear log.level --INFO --openflow.flow_tracker=WARNING --openflow.igmp_manager=DEBUG --openflow.groupflow=DEBUG –openflow=DEBUG
```

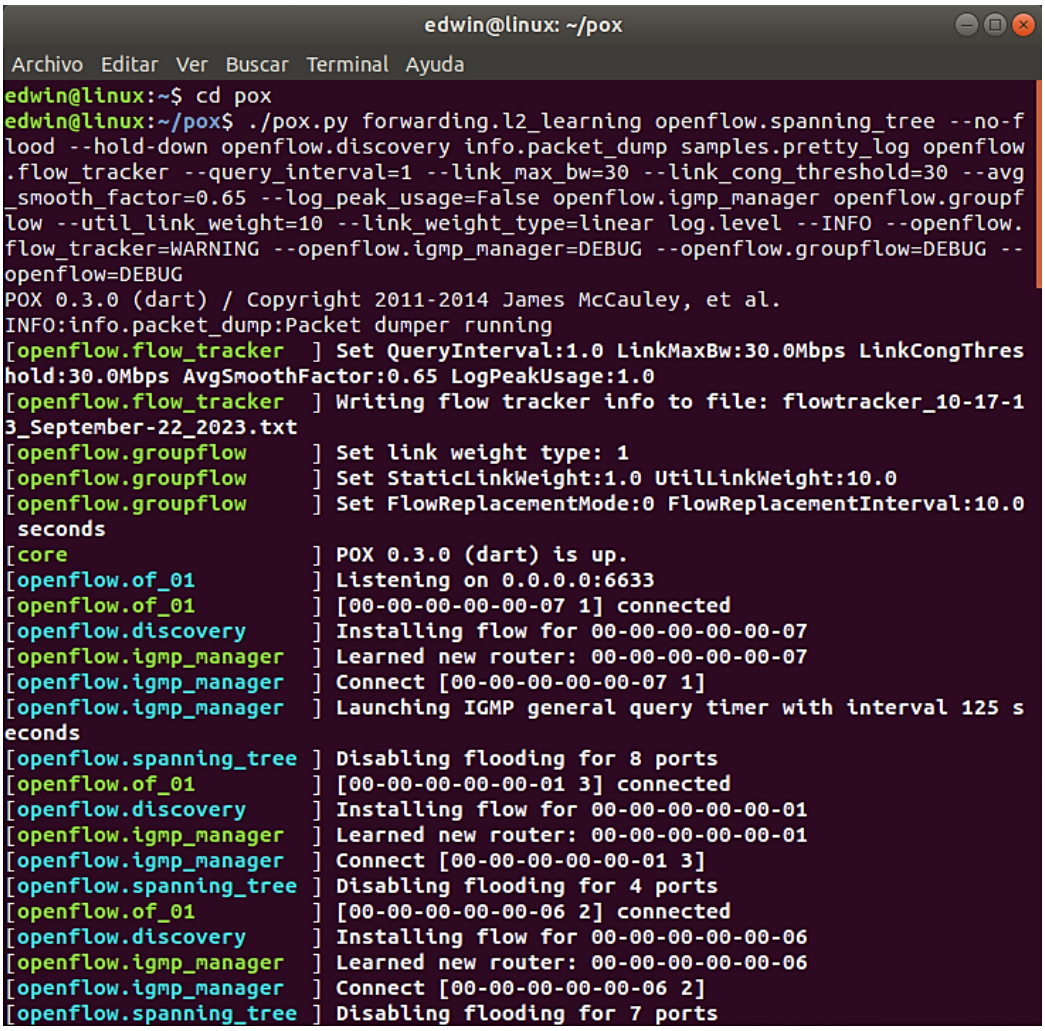


Figura 29. *Implementación del módulo GroupFlow en el entorno POX.*
Fuente: *Elaborado por el Autor.*

3.2.4 DESARROLLO DE ESCENARIOS DE RED SDN EN MININET UTILIZANDO PYTHON

Se implementaron tres topologías de red distintas: lineal, árbol y malla, utilizando el emulador de red MiniNet. Para este fin, se ha optado por el lenguaje de programación Python. Se ha elaborado un script específico para cada topología de red, así como para la implementación de los controladores de red OpenDayLight y POX. A continuación, se ofrece una descripción detallada del proceso de codificación llevado a cabo en un escenario de topología de red lineal con el controlador ODL. La codificación completa realizada para las topologías árbol y malla con ODL y POX se encuentra en la sección de anexos.

Inicialmente, se hizo uso de la biblioteca de MiniNet para importar y administrar las librerías que contienen las funciones requeridas para el desarrollo de los escenarios de red SDN.

Inicializar topología

Se define una clase llamada 'SDN_Topo' que hereda de la clase 'Topo' de la biblioteca MiniNet y un método especial '__init__' que se ejecuta cuando se crea una instancia de la clase 'SDN_Topo'. Este método llama al método '__init__' de la clase padre 'Topo' para inicializar la topología de red.

Añadir hosts y switches

Se añaden hosts y switches a la topología usando el método 'addHost' y el método 'addSwitch' de la clase 'Topo'. Cada host y switch tiene un nombre único como argumento, y los hosts también tienen una dirección IP asignada. Los switches son de tipo 'OVSSwitch' ya que se usan switches virtuales Open vSwitch.

Añadir enlaces (links)

El método ‘addLink’ de la clase ‘Topo’ permite la adición de enlaces de red. Cada enlace tiene un ancho de banda (BW) en Mbps y un parámetro booleano ‘use_htb’, que indica que se utiliza el controlador de ancho de banda HTB (Hierarchical Token Bucket) para controlar la congestión de la red. Además, este método permite asignar un valor de retardo en “ms” y un valor de pérdida de paquetes en “%”. La topología creada por esta clase consta de 9 hosts cliente y 3 switches, siguiendo el esquema de diseño para una topología de red lineal. Los hosts serv1, serv2 y serv3 desempeñan el papel de servidores de streaming y están enlazados a los switches S1, S2 y S3, respectivamente. Los scripts desarrollados abarcan tanto los escenarios en los que se mantuvieron los parámetros de la topología de red en sus valores por defecto, como aquellos en los que se introdujeron variaciones en los parámetros de los enlaces de la topología de red.

Configuración de red para usar el controlador externo ODL o POX.

Se define una función llamada 'SDN_Config' que recibe tres argumentos: 'topo', 'hosts' e 'interactive'. Se crea una instancia de la clase MiniNet con el argumento 'topo', que es una instancia de la clase 'SDN_Topo' que representa la topología de red. La instancia 'RemoteController' de MiniNet permite comunicarse con un controlador externo mediante el protocolo OpenFlow que se ejecuta en la dirección IP localhost '127.0.0.1', mediante el puerto '6633' y utiliza switches virtuales Open vSwitch.

Configuración de interfaces de control en los switches de red

Se emplea el método 'get' de la clase MiniNet para obtener el nombre de cada switch y el método 'cmd' de la clase 'switch' para ejecutar comandos en el switch. Se añade una ruta en la tabla de enrutamiento del switch que indica que los paquetes dirigidos a la dirección '127.0.0.1' se deben enviar por la interfaz local. También se añade una ruta en la tabla de enrutamiento del controlador que indica que los paquetes dirigidos a la dirección IP del switch se deben enviar por la interfaz local del controlador. En resumen, se configura la interfaz de control en los switches de red y se agrega rutas para cada host.

Configuración de hosts para soporte multicast con IGMPv3 en Python

Para permitir el envío y la recepción de paquetes multicast, que son paquetes que se envían a un grupo de multidifusión en lugar de a uno solo, se configuran los hosts de la red. Para ello, se configuran las rutas multicast para interfaces virtuales. La función “net.get(host_name).cmd()” ejecuta un comando en la terminal del host especificado por “host_name”. En este caso, el comando es “route add -net 224.0.0.0/4 ' + host_name + '-eth0'”, que agrega una ruta multicast para la interfaz virtual “eth0” del host. Además, se establece la versión de IGMP en la versión 3 para todas las interfaces de red del host desde el kernel de Linux mediante el comando “echo 3 > /proc/sys/net/ipv4/conf/all/force_igmp_version”.

La configuración mencionada se lleva a cabo únicamente en un escenario que involucra el controlador de red ODL, ya que el controlador POX utiliza el módulo GroupFlow para permitir la multidifusión con IGMPv3.

Permitir tiempo para que el controlador detecte la topología de red

Se configura un tiempo de espera determinado (10 segundos) para permitir que el controlador remoto detecte la topología de la red y establezca las reglas de reenvío en los switches mediante el protocolo OpenFlow. Se lanza una interfaz de línea de comandos (CLI) para interactuar con la red virtual, usando el método CLI de la clase Mininet. Esta interfaz permite ejecutar comandos en los hosts y switches de la red, así como en el controlador remoto. Posteriormente se detiene la instancia de Mininet, lo que elimina la red virtual y libera los recursos usados.

Generar topología y ejecutar escenario SDN

Finalmente, se implementan las funciones que dan forma a la topología de red y se inicia el escenario de red con las configuraciones previamente definidas en MiniNet.

El código completo, desarrollado en el lenguaje de programación Python para una topología de red lineal con el controlador de red OpenDaylight (ODL), se presenta a continuación. Los scripts completos correspondientes a los escenarios de red para una topología de red árbol y malla se encuentran en la sección de anexos.

Tabla 14. *Script que incorpora una topología de red lineal con ODL.*
Fuente: *Elaborado por el Autor.*

```
1.  #!/usr/bin/env python
2.  # coding=utf-8
3.  from mininet.net import Mininet
4.  from mininet.topo import *
5.  from mininet.net import *
6.  from mininet.node import OVSSwitch, UserSwitch
7.  from mininet.link import TCLink
8.  from mininet.log import setLogLevel
9.  from mininet.cli import CLI
10. from mininet.node import Node, RemoteController
11. from time import sleep, time
12. from datetime import datetime
13. from subprocess import call
14.
15.
16. class SDN_Topo( Topo ):
17.     def __init__( self ):
18.         # Inicializar topología
19.         Topo.__init__( self )
20.
21.         # Añadir hosts y switches
22.         h1 = self.addHost('h1', ip='10.0.0.1')
23.         h2 = self.addHost('h2', ip='10.0.0.2')
24.         h3 = self.addHost('h3', ip='10.0.0.3')
25.         h4 = self.addHost('h4', ip='10.0.0.4')
26.         h5 = self.addHost('h5', ip='10.0.0.5')
27.         h6 = self.addHost('h6', ip='10.0.0.6')
28.         h7 = self.addHost('h7', ip='10.0.0.7')
29.         h8 = self.addHost('h8', ip='10.0.0.8')
30.         h9 = self.addHost('h9', ip='10.0.0.9')
31.         serv1 = self.addHost('serv1', ip='10.1.1.1')
32.         serv2 = self.addHost('serv2', ip='10.1.1.2')
33.         serv3 = self.addHost('serv3', ip='10.1.1.3')
34.
35.         s1 = self.addSwitch('s1')
36.         s2 = self.addSwitch('s2')
```

```
37.         s3 = self.addSwitch('s3')
38.
39.
40.         # Añadir enlaces (links)
41.         self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
42.         self.addLink(s2, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
43.         self.addLink(s1, h1, bw = 1000, use_htb = True)
44.         self.addLink(s1, h2, bw = 1000, use_htb = True)
45.         self.addLink(s1, h3, bw = 1000, use_htb = True)
46.         self.addLink(s2, h4, bw = 1000, use_htb = True)
47.         self.addLink(s2, h5, bw = 1000, use_htb = True)
48.         self.addLink(s2, h6, bw = 1000, use_htb = True)
49.         self.addLink(s3, h7, bw = 1000, use_htb = True)
50.         self.addLink(s3, h8, bw = 1000, use_htb = True)
51.         self.addLink(s3, h9, bw = 1000, use_htb = True)
52.         self.addLink(serv1, s1, bw = 1000, use_htb = True)
53.         self.addLink(serv2, s2, bw = 1000, use_htb = True)
54.         self.addLink(serv3, s3, bw = 1000, use_htb = True)
55.
56.
57.     def get_host_list(self):
58.         return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
    'serv3']
59.
60.     def get_switch_list(self):
61.         return ['s1', 's2', 's3']
62.
63.
64. def SDN_Config(topo, hosts = [], interactive = False):
65.
66.     # Configuración de red para usar el controlador externo ODL.
67.     net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
    link=TCLink, build=False, autoSetMacs=True)
68.     net.addController('odl', RemoteController, ip = '127.0.0.1', port = 6633)
69.     net.start()
70.
71.     # Configurar interface de control en switches de red
72.     for switch_name in topo.get_switch_list():
73.         net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
74.         net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
75.         net.get('odl').cmd('route add -host ' + net.get(switch_name).IP() + '
    dev lo')
76.
77.     # Configuración de hosts para soporte multicast con igmpv3
78.     for host_name in topo.get_host_list():
79.         # Configurar rutas multicast para interfaces virtuales
80.         net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
    eth0')
```

```
        # Configurar igmpv3 en cada host
        net.get(host_name).cmd('echo 3 > /proc/sys/net/ipv4/conf/all/force_igmp_version')

    # Permitir tiempo para que el controlador ODL detecte la topología
    sleep_time = 10
    print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para permitir al controlador ODL descubrir la topología de red\033[0m'
    sleep(sleep_time)

    print("\033[1;36m Red SDN con una Topología Lineal para Multidifusión con ODL\033[0m")

    CLI(net)
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )
    # Generar topología y ejecutar escenario SDN
    topo = SDN_Topo()
    hosts = topo.get_host_list()
        SDN_Config(topo, hosts, False)
```

3.2.5 STREAMING UTILIZANDO DIFERENTES PROTOCOLOS DE TRANSMISIÓN

En cada escenario de red, se implementó un servicio de streaming adaptado a tres topologías de red distintas en MiniNet: lineal, árbol y malla. Se utilizaron los protocolos de transmisión de streaming RTMP, RTSP, RTP y UDP. Para la gestión de la red en cada escenario, se empleó un controlador SDN, siendo OpenDaylight y POX los seleccionados para esta tarea.

Los escenarios desarrollados se localizan en el directorio denominado “custom”, el cual se encuentra dentro del directorio principal de MiniNet. Se pueden acceder a estos a través de una ventana de terminal utilizando la Interfaz de Línea de Comandos (CLI). Los scripts desarrollados pueden ser ejecutados utilizando el comando “sudo python [nombre de archivo].py”. Cada script tiene la finalidad de implementar una topología de red SDN en MiniNet con capacidad para realizar multidifusión, para lo cual utiliza un controlador SDN remoto que se ejecuta en la dirección IP 127.0.0.1 con puerto 6633, ya sea ODL o POX.

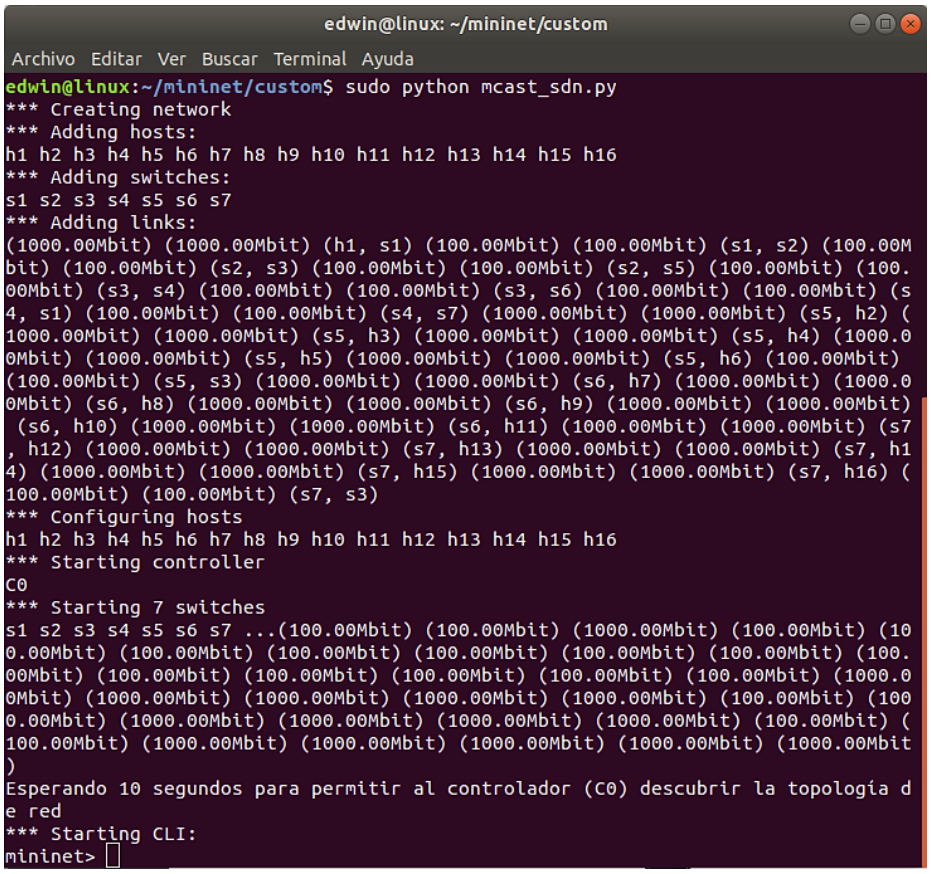


Figura 30. *Implementación de topología de red malla en el entorno de MiniNet.*
Fuente: *Elaborado por el Autor.*

Para la transmisión en tiempo real (streaming), se utilizó un archivo multimedia en formato MP4, denominado “stream.mp4”. Este archivo está disponible y se puede acceder desde los servidores de streaming: serv1, serv2 y serv3 en la red de MiniNet.

Tabla 15. *Propiedades de archivo stream.mp4*
Fuente: *Elaborado por el Autor.*

Propiedades	Detalles
Nombre:	stream
Formato:	MP4
Códec:	H.264
Tamaño:	158MB
Dimensiones:	1280 x 720
Duración:	42 minutos:28 segundos
Tasa de fotogramas:	25 fotogramas por segundo

STREAMING CON EL PROTOCOLO RTMP:

El protocolo RTMP opera sobre TCP, lo que intrínsecamente impide la posibilidad de realizar transmisiones en multidifusión. Por lo tanto, se llevó a cabo una

transmisión de video en streaming en Unicast. Para configurar el servidor de streaming se utilizó el comando 'xterm [nombre de servidor]', que abre la Interfaz de Línea de Comandos (CLI). Para la transmisión de video en streaming, se emplea el servidor mediamtx y el software FFmpeg utilizando los siguientes comandos:

```
> ffmpeg -re -stream_loop -1 -i Escritorio/stream.mp4 -c copy -f flv rtmp://10.0.0.1/live
```

Tabla 16. *Descripción de instrucciones utilizadas en FFmpeg.*
Fuente: *Extraído del sitio web de FFmpeg.*

Comando	Descripción
ffmpeg	Comando principal que inicia el programa FFmpeg.
-re	FFmpeg lee el archivo de entrada a su velocidad nativa (la velocidad a la que se grabó o se supone que se debe reproducir). Sin este comando, FFmpeg leería el archivo de entrada lo más rápido posible.
-stream_loop -1	Este comando hace que ffmpeg repita el stream de entrada indefinidamente. El -1 significa que no hay límite en el número de repeticiones.
-i Escritorio/stream.mp4	Este es el archivo de entrada que ffmpeg va a leer. En este caso, es un archivo llamado stream.mp4 en el directorio Escritorio.
-c copy	Esto hace que FFmpeg copie los streams de entrada tal como están, sin tratar de convertirlos o recodificarlos.
-f flv	Este comando especifica el formato de salida. En este caso, es flv, que es un formato de archivo utilizado para entregar video a través de Internet utilizando Adobe Flash Player.
rtmp://10.0.0.1/live	Esta es la dirección IP del servidor desde donde FFmpeg enviará el stream de medios. En este caso, es una dirección RTMP, que es un protocolo utilizado para transmitir audio, video y datos a través de Internet.

FFmpeg realiza la publicación de la transmisión en el servidor MediaMTX, el cual envía la transmisión a los clientes finales mediante el uso del protocolo RTMP.

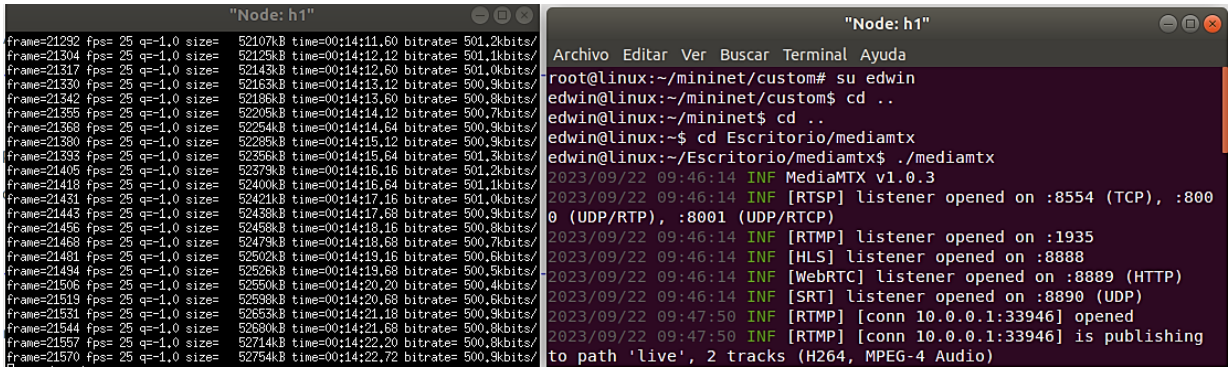


Figura 31. *Servidor MediaMTX con compatibilidad para RTMP.*
Fuente: *Elaborado por el Autor.*

Las estaciones cliente, identificadas como h1-h9 de cada escenario de red, tienen la capacidad de acceder a la transmisión de streaming en vivo mediante el uso del reproductor multimedia VLC y del protocolo RTMP (Real Time Messaging Protocol). Para ello, deben dirigirse a la dirección IP del servidor de streaming y utilizar el puerto '1935'.

Tabla 17. *Instrucción para la recepción de streaming con RTMP en VLC.*
Fuente: *Extraído del sitio web de VideoLAN Wiki.*

Instrucción	Descripción
vlc rtmp://[IP de servidor de streaming]:1935/live	Este comando le dice a VLC que reproduzca un flujo de medios en vivo, desde un servidor en la dirección IP del servidor de streaming utilizando el puerto '1935' y el protocolo RTMP.

STREAMING MULTICAST CON LOS PROTOCOLO RTSP, RTP Y UDP:

Se ha establecido un servicio de transmisión de video en tiempo real utilizando el protocolo RTSP. Este protocolo opera en sinergia con los protocolos RTP y UDP, lo que facilita la realización de una multidifusión a una dirección IP multicast, también conocido como grupo de multidifusión. Aunque es factible realizar la multidifusión exclusivamente a través de RTP o UDP, el protocolo RTSP no puede soportar la multidifusión de manera independiente, ya que requiere un protocolo de transporte no orientado a conexión para cumplir con este propósito. En un terminal de línea de comandos, se configura el servidor de transmisión de

video en tiempo real utilizando el software VLC y el protocolo RTSP a través de los siguientes comandos:

```
> su -c 'vlc --repeat Escritorio/stream.mp4 -I dummy --sout "#rtp{access=udp, mux=ts, proto=udp, dst=224.2.0.1, port=3456, sdp=rtsp://[IP_servidor]:8554/live, ttl=16}'" edwin
```

Una alternativa para realizar multidifusión es mediante el uso del protocolo RTP, el cual opera sobre UDP. A continuación, se presenta la configuración del servidor de transmisión de video utilizando este protocolo mediante VLC:

```
> su -c 'vlc --repeat Escritorio/stream.mp4 -I dummy --sout "#rtp{access=udp, mux=ts, proto=udp, dst=224.1.1.1, port=5007 ttl=16}'" edwin
```

Configuración del servidor para la transmisión de video en streaming únicamente a través del protocolo UDP:

```
> su -c 'vlc --repeat Escritorio/stream.mp4 -I dummy --sout "#udp{dst=239.0.0.1, ttl=16}'" edwin
```

Tabla 18. *Instrucciones en VLC para la transmisión de streaming con RTSP.*
Fuente: *Extraído del sito web VideoLAN Wiki.*

Comando	Descripción
su -c	Se utiliza para cambiar a un usuario específico y ejecuta los comandos que siguen después de 'su -c'.
--repeat	Indica que el archivo de video se reproducirá en bucle.
Escritorio/stream.mp4	Ruta al archivo de video que se va a transmitir.
-I dummy	Ejecuta VLC sin interfaz gráfica de usuario.
access=udp	Utiliza el protocolo UDP para la transmisión.
mux=ts	Utiliza MPEG-TS (Transport Stream) para multiplexar el video y el audio.
proto=udp	Indica que el protocolo de transmisión es UDP.
dst=224.2.0.1	Esta es la dirección IP de destino para multidifusión.

port=3456	Este es el puerto de destino para multidifusión.
sdp=rtsp://[IP_servidor]:8554/live	Esta es la URL del Protocolo de Descripción de Sesión (SDP) que describe los parámetros multimedia. Utiliza el protocolo RTSP con la dirección IP y puerto correspondiente.
ttl=16	Este es el tiempo de vida (TTL) para los paquetes multicast.

Las estaciones cliente, identificadas como “h1-h9” de cada escenario de red, tienen la capacidad de acceder a la transmisión de video en tiempo real al unirse al grupo de multidifusión a través de la dirección IP multicast correspondiente. Para lograr este propósito, se detallan a continuación los comandos utilizados con el reproductor multimedia VLC:

Tabla 19. *Instrucciones para la recepción de streaming multicast con VLC.*
Fuente: *Elaborado por el Autor.*

Protocolo	Instrucciones	Descripción
RTSP	vlc rtsp://[IP_servidor]:8554/live?vlcmulticast	Abre VLC y se conecta a una transmisión de video en vivo a través de RTSP en la dirección IP del servidor de streaming utilizando el puerto ‘8554’. El parámetro ‘vlcmulticast’ indica que la transmisión es de tipo multicast.
RTP	vlc rtp://224.1.1.1:5007/live	Este comando abre VLC y se conecta a una transmisión de video en vivo a través de RTP (Real-time Transport Protocol) en la dirección ‘224.1.1.1’ en el puerto ‘5007’.
UDP	vlc udp://@239.0.0.1	Este comando abre VLC y se conecta a una transmisión de video en vivo a través de UDP (User Datagram Protocol) en la dirección ‘224.2.0.1’. El ‘@’ antes de la dirección indica que VLC debe unirse al grupo multicast en esa dirección.

3.3 ESTUDIO DE FACTIBILIDAD

FACTIBILIDAD TÉCNICA

Actualmente, el despliegue de las Redes Definidas por Software (SDN) se encuentra en una fase de desarrollo. Esta tecnología emergente promete revolucionar la forma en que se diseñan, implementan y administran las redes de telecomunicaciones. Sin embargo, para el despliegue efectivo de soluciones basadas en SDN, es imprescindible contar con recursos técnicos adecuados. Esto incluye tanto el software como el hardware necesarios. A medida que esta tecnología continúa evolucionando, es probable que veamos cambios significativos en el panorama de las telecomunicaciones.

Para el desarrollo de los escenarios de simulación de Redes Definidas por Software (SDN), se realizó un análisis de las tecnologías apropiadas, los dispositivos de red y los protocolos de comunicación necesarios. Este análisis consideró los requisitos y aspectos esenciales para asegurar que los escenarios de red sean diseñados, desarrollados y configurados de manera adecuada, utilizando las tecnologías y protocolos más apropiados para la transmisión de datos.

La creación de los escenarios de simulación para las Redes Definidas por Software (SDN) se llevó a cabo siguiendo el modelo de arquitectura específico para las redes SDN. Para este propósito, se necesitaron varios recursos técnicos. Entre ellos, los controladores de red SDN, que son esenciales para la gestión y el control del comportamiento de la red. Además, se utilizó el software de emulación de redes MiniNet, este software tiene la capacidad de emular desde switches con soporte para OpenFlow hasta enlaces y hosts virtuales.

Los recursos de software utilizados para el entorno de emulación de Redes Definidas por Software (SDN) en el desarrollo del presente proyecto son de código abierto y se encuentran disponibles de manera gratuita. No obstante, la implementación de los escenarios de red puede llevarse a cabo en un entorno de red real con switches SDN de hardware. Esta posibilidad facilita de manera significativa el despliegue y desarrollo de los escenarios de simulación propuestos, permitiendo un control más eficaz y eficiente de los recursos de red.

ESTUDIO DE COSTOS COMPARATIVO

Para este análisis se estimaron los costos asociados con el desarrollo de los escenarios de simulación de Redes Definidas por Software (SDN), tanto en un entorno de simulación como en un entorno real con dispositivos de hardware.

COSTOS EN UN ENTORNO DE EMULACIÓN DE RED:

Costes de controladores de red SDN:

Tabla 20*.Costos de controladores de red SDN.*
Fuente: *Elaborado por el Autor.*

Cant.	Descripción	P. Total
1	Controlador de red SDN OpenDayLight Oxygen karaf-0.8.4	$0
1	Controlador de red SDN POX 0.3.0 dart.	$0
	TOTAL	$0

Costos de equipos y recursos de software en un entorno de simulación:

Tabla 21. *Costos de recursos en un entorno emulado.*
Fuente: *Elaborado por el Autor.*

Cant.	Descripción	P. Total
1	MiniNet 2.3.1b1	$0
1	Open vSwitch 2.9.8	$0
1	Wireshark 2.6.10	$0
1	Reproductor multimedia VLC 3.0.8 Vetinari	$0
1	Reproductor multimedia Ffmpeg N-109444-geef763c705	$0
1	Sistema Operativo Ubuntu 18.04.6 LTS	$0
1	Computadora DELL Inspiron 3442	$450
	TOTAL	$450

COSTOS EN UN ENTORNO DE RED REAL:

Para la implementación de los escenarios de red en un entorno real, es esencial considerar los costos asociados a los componentes que se emulan en MiniNet para cada escenario, teniendo en cuenta su respectiva topología de red. A continuación, se proporciona un desglose detallado de los costos asociados para su consideración.

Costos de implementación de escenario con topología de red lineal.

Tabla 22. *Costos de implementación de escenario con topología de red lineal.*
Fuente: *Elaborado por el Autor.*

Cant.	Descripción	P. Unitario	P. Total
3	Switch SDN de Hardware SEL-2742S	$2.790	$8.370
12	Computadoras de escritorio	$450	$5.400
1	Rollo de 50 metros de Cable UTP Cat6	$38	$38
28	Conectores RJ45	$0,17	$4,76
28	Protectores plásticos para RJ45	$0,20	$5,60
		TOTAL	$13.818,36

Costos de implementación de escenario con topología de red árbol.

Tabla 23. *Costos de implementación de escenario con topología de red árbol.*
Fuente: *Elaborado por el Autor.*

Cant.	Descripción	P. Unitario	P. Total
7	Switch SDN de Hardware SEL-2742S	$2.790	$19.530
12	Computadoras de escritorio	$450	$5.400
1	Rollo de 50 metros de Cable UTP Cat6	$38	$38
36	Conectores RJ45	$0,17	$6,12
36	Protectores plásticos para RJ45	$0,20	$7,20
		TOTAL	$24.801,32

Costos de implementación de escenario con topología de red malla

Tabla 24. *Costos de implementación de escenario con topología de red malla.*
Fuente: *Elaborado por el Autor.*

Cant.	Descripción	P. Unitario	P. Total
7	Switch SDN de Hardware SEL-2742S	$2.790	$19.530
12	Computadoras de escritorio	$450	$5.400
1	Rollo de 50 metros de Cable UTP Cat6	$38	$38
48	Conectores RJ45	$0,17	$8,16
48	Protectores plásticos para RJ45	$0,20	$9,60
		TOTAL	$24.985,76

COMPARATIVA DEL COSTO TOTAL EN UN ENTORNO EMULADO Y REAL

En la Tabla 25 se muestra una comparación del costo asociado a cada componente requerido para la implementación de los escenarios de red SDN, tanto en un entorno de red emulado como un entorno de red real.

Tabla 25. *Comparativa del costo total en un entorno emulado y real.*
Fuente: *Elaborado por el Autor.*

Entorno de red	Descripción	P. Total
Emulado	Escenarios de red emulados en MiniNet	$450
Real	Escenario de red SDN con topología lineal	$13.818,36
	Escenario de red SDN con topología árbol	$24.801,32
	Escenario de red SDN con topología malla	$24.985,76
	TOTAL	$64.055,44

3.4 RESULTADOS

PRUEBA DE CONECTIVIDAD DE RED SDN EN MININET

Para la implementación de un servicio de transmisión en streaming sobre la infraestructura de Red Definida por Software (SDN) en diferentes topologías de red, se requirió inicialmente verificar la existencia de conectividad y la accesibilidad de todos los anfitriones en la red. Para lograr esto, se empleó la Interfaz de Línea de Comandos (CLI) del software MiniNet y se ejecutó una prueba de conectividad a todos los anfitriones de la red en cada escenario utilizando el comando “pingall”.

Se llevó a cabo un ensayo de conectividad en un escenario que presenta una topología malla. Este ensayo se realizó utilizando el controlador OpenDaylight (ODL), y se aplicaron los parámetros predeterminados de la topología de red.

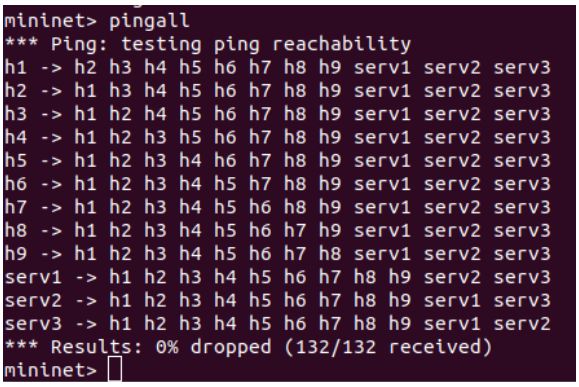


Figura 32. *Prueba de conectividad en escenario con parámetros de red por defecto.*
Fuente: *Elaborado por el Autor.*

Se efectuó una prueba de conectividad en un escenario que cuenta con una topología malla. Este examen se realizó utilizando el controlador OpenDaylight (ODL), y se introdujeron variaciones en los parámetros de enlace de la topología de red, tal como se especifica en la Tabla 10.

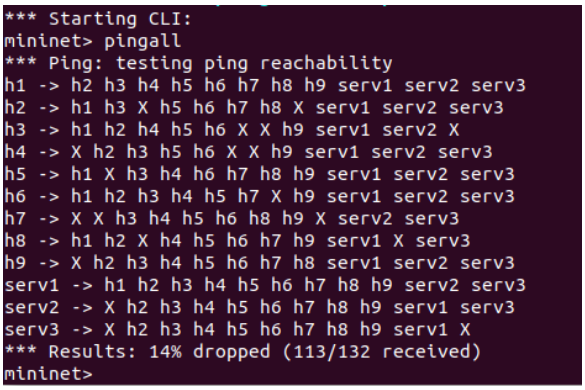


Figura 33. *Prueba de conectividad en escenario con adición de tasa de pérdida y tiempo de retardo.*
Fuente: *Elaborado por el Autor.*

RESULTADOS DE STREAMING CON EL PROTOCOLO RTMP

El host identificado como “serv1”, transmite un archivo multimedia en formato MP4, denominado “stream.mp4” a través de la red definida por software (SDN) utilizando el protocolo de mensajería en tiempo real (RTMP). Esta transmisión se dirige a diversas estaciones cliente y sus direcciones IP se detallan en la Tabla 26. La red es gestionada por un controlador de red remoto que utiliza el protocolo OpenFlow para su administración. Cada estación cliente accede a la transmisión mediante el uso del reproductor multimedia VLC.

Tabla 26. *Direcciones IP de nodos en escenario con RTMP.*
Fuente: *Elaborado por el Autor.*

Servidor de streaming			Estaciones cliente	
Host	**IP**	**Puerto**	**Host**	**IP**
serv1	10.1.1.1	1935	h1	10.0.0.1
			h2	10.0.0.2
			h3	10.0.0.3
			h4	10.0.0.4
			h5	10.0.0.5
			h6	10.0.0.6
			h7	10.0.0.7
			h8	10.0.0.8
			h9	10.0.0.9

Para la implementación del protocolo de mensajería en tiempo real (RTMP), se requirió el uso del servidor MediaMTX, que proporciona soporte para la publicación y transmisión de contenido a través del protocolo RTMP. Además, se hizo uso del software FFmpeg en este proceso de simulación.

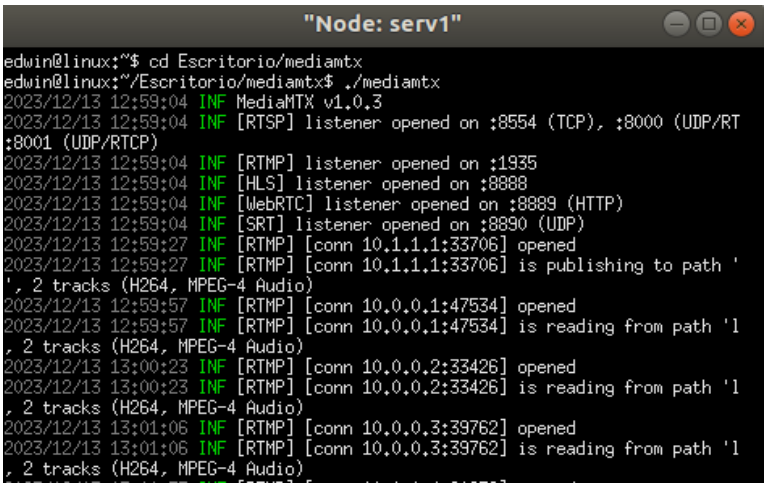


Figura 34. *MediaMTX, publicación y transmisión de contenido con RTMP.*
Fuente: *Elaborado por el Autor.*

En este escenario de simulación, se llevó a cabo la visualización de la transmisión en tiempo real a través de la dirección IP “10.1.1.1”, utilizando el puerto “1935” y el protocolo de mensajería en tiempo real (RTMP). Para ello, se utilizó el reproductor multimedia VLC, que está integrado en las estaciones cliente.

Figura 35. *Visualización de streaming con el protocolo RTMP.*
Fuente: *Elaborado por el Autor.*

En el software de análisis de protocolos Wireshark, se realiza una captura de paquetes con la finalidad de identificar la presencia del protocolo RTMP y el protocolo de transporte TCP dentro de los paquetes que han sido capturados.

Source	Destination	Protocol	Length	dst port	Info
10.1.1.1	10.0.0.7	RTMP	115	50382	Video Data
10.0.0.7	10.1.1.1	TCP	66	1935	50382 → 1935 [ACK]
10.1.1.1	10.0.0.7	RTMP	417	50382	User Control Messag
10.0.0.7	10.1.1.1	TCP	66	1935	50382 → 1935 [ACK]
10.1.1.1	10.0.0.7	RTMP	141	50382	Video Data
10.0.0.7	10.1.1.1	TCP	66	1935	50382 → 1935 [ACK]

Figura 36*. RTMP, S7-eth4*
Fuente: *Elaborado por el Autor.*

RESULTADOS DE STREAMING CON EL PROTOCOLO RTSP, RTP Y UDP

Los host serv1, serv2 y serv3 realizan la transmisión de un archivo multimedia denominado “stream.mp4”, a una dirección de multidifusión a través de la Red Definida por Software (SDN) utilizando el Protocolo de Streaming en Tiempo Real (RTSP). Esta transmisión se dirige a diversas estaciones cliente, cuyas direcciones IP se detallan en la Tabla 27. La red es administrada por un controlador de red remoto que utiliza el protocolo OpenFlow para su gestión. Cada estación cliente realiza una petición para unirse al grupo multicast específico, con el objetivo de acceder a la transmisión en tiempo real. Esta operación se lleva a cabo mediante el uso del reproductor multimedia VLC.

Tabla 27. *Direcciones IP de nodos para streaming multicast.*
Fuente: *Elaborado por el Autor.*

Servidores de streaming		Protocolo	Grupos multicast		Estaciones cliente	
Host	**IP**		**IP**	**Puerto**	**Host**	**IP**
serv1	10.1.1.1	RTSP	224.2.0.1	3456	h1	10.0.0.1
					h2	10.0.0.2
					h3	10.0.03
serv2	10.1.1.2	RTP	224.1.1.1	5007	h4	10.0.0.4
					h5	10.0.0.5
					h6	10.0.0.6
serv3	10.1.1.3	UDP	239.0.0.1	1234	h7	10.0.0.7
					h8	10.0.0.8
					h9	10.0.0.9

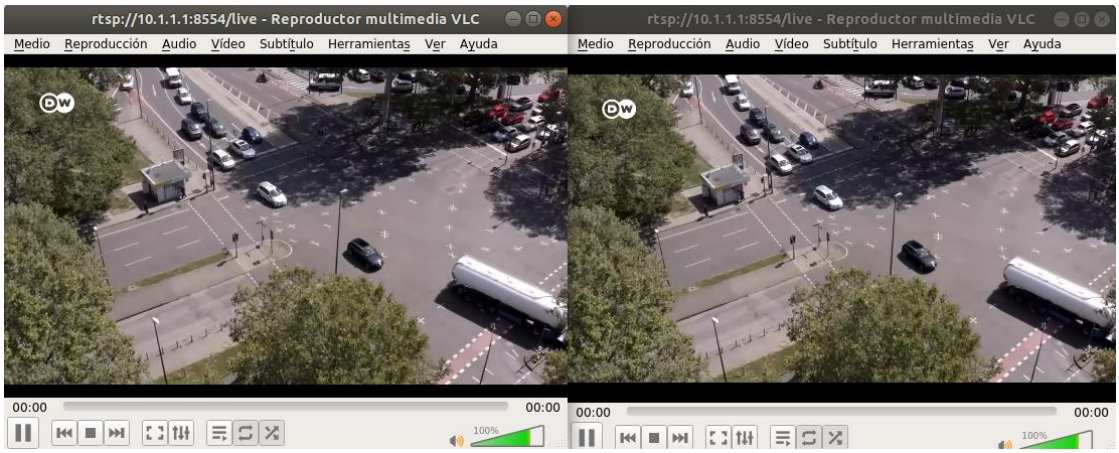


Figura 37. *Visualización de streaming multicast con el protocolo RTSP.*
Fuente: *Elaborado por el Autor.*

Figura 38. *Visualización de streaming multicast con el Protocolo RTP.*
Fuente: *Elaborado por el Autor.*

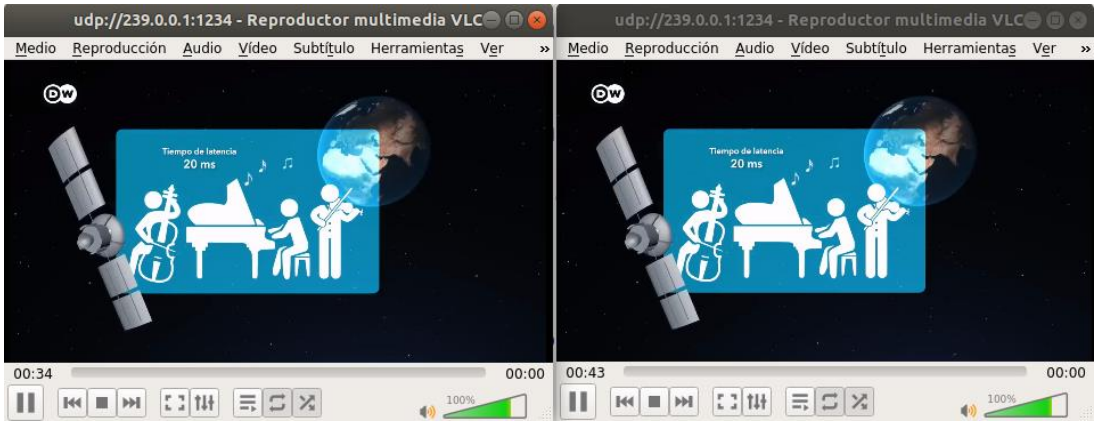


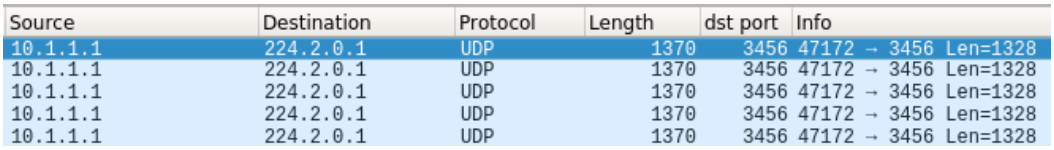
Figura 39. *Visualización de streaming multicast con el Protocolo UDP.*
Fuente: *Elaborado por el Autor.*

En cada terminal de cliente final, se puede visualizar la transmisión multicast en tiempo real (streaming) mediante el uso del reproductor multimedia VLC.

Se utiliza del software de análisis de protocolos Wireshark, para llevar a cabo una operación de captura de paquetes. El objetivo principal de esta operación es la identificación precisa de las direcciones IP que corresponden a los grupos multicast, así como la detección del protocolo RTSP y el protocolo de transporte UDP. Todos estos elementos se encuentran presentes en los paquetes que han sido capturados durante la transmisión en streaming.

Source	Destination	Protocol	Length	dst port	Info
10.0.0.3	10.1.1.1	RTSP	211	8554	GET_PARAMETER rtsp://10.1.1.1:8554/live RTSP/1.0
10.1.1.1	10.0.0.3	RTSP	225	42104	Reply: RTSP/1.0 200 OK

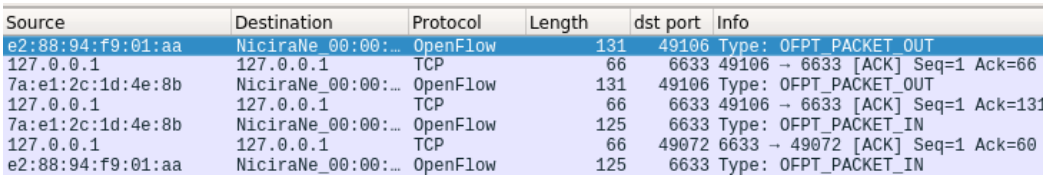
Figura 40. *RTSP, S5-eth6.*
Fuente: *Elaborado por el Autor.*



Source	Destination	Protocol	Length	dst port	Info
10.1.1.1	224.2.0.1	UDP	1370	3456	47172 → 3456 Len=1328
10.1.1.1	224.2.0.1	UDP	1370	3456	47172 → 3456 Len=1328
10.1.1.1	224.2.0.1	UDP	1370	3456	47172 → 3456 Len=1328
10.1.1.1	224.2.0.1	UDP	1370	3456	47172 → 3456 Len=1328
10.1.1.1	224.2.0.1	UDP	1370	3456	47172 → 3456 Len=1328

Figura 41. *UDP, S5-eth6.*
Fuente: *Elaborado por el Autor.*

Además, se pudo identificar la presencia del protocolo OpenFlow en los paquetes de red que fueron capturados durante el proceso de transmisión en streaming.



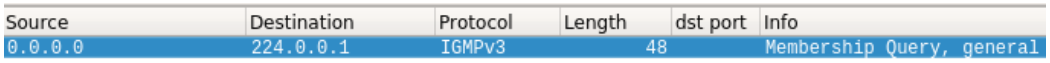
Source	Destination	Protocol	Length	dst port	Info
e2:88:94:f9:01:aa	NiciraNe_00:00:…	OpenFlow	131	49106	Type: OFPT_PACKET_OUT
127.0.0.1	127.0.0.1	TCP	66	6633	49106 → 6633 [ACK] Seq=1 Ack=66
7a:e1:2c:1d:4e:8b	NiciraNe_00:00:…	OpenFlow	131	49106	Type: OFPT_PACKET_OUT
127.0.0.1	127.0.0.1	TCP	66	6633	49106 → 6633 [ACK] Seq=1 Ack=131
7a:e1:2c:1d:4e:8b	NiciraNe_00:00:…	OpenFlow	125	6633	Type: OFPT_PACKET_IN
127.0.0.1	127.0.0.1	TCP	66	49072	6633 → 49072 [ACK] Seq=1 Ack=60
e2:88:94:f9:01:aa	NiciraNe_00:00:…	OpenFlow	125	6633	Type: OFPT_PACKET_IN

Figura 42. *Protocolo OpenFlow, Loopback.*
Fuente: *Elaborado por el Autor.*

UNIÓN Y ABANDONO DE GRUPOS DE MULTIDIFUSIÓN CON EL PROTOCOLO IGMPv3:

Durante la transmisión en streaming en los escenarios de red, se llevó a cabo una captura de los paquetes multicast. El objetivo principal de esta operación era la identificación de los diversos tipos de mensajes IGMPv3 que se hallaban presentes en los paquetes capturados, los cuales fueron registrados mediante el uso del software Wireshark.

Para los escenarios de red, contamos con tres grupos de multidifusión. A estos grupos se encuentran conectados diversas estaciones cliente, las cuales reciben la transmisión en streaming por medio de VLC acorde a las especificaciones delineadas en la Tabla 27.

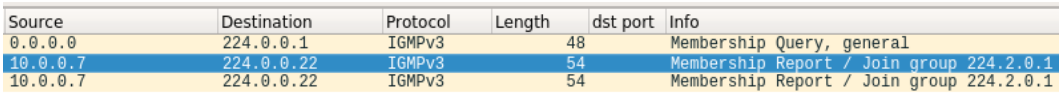


Source	Destination	Protocol	Length	dst port	Info
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, general

Figura 43. *IGMPv3 Membership general Query, S7-eth4.*
Fuente: *Elaborado por el Autor.*

Cuando un nodo toma la decisión de incorporarse a la transmisión, es imperativo que envíe un mensaje clasificado como “Join group” para poder acceder a la transmisión perteneciente al grupo de multidifusión específico. En este caso en particular, procedemos a integrar el host “h7” con dirección IP “10.0.0.7” a la dirección de grupo “224.2.0.1”. Posteriormente, realizamos una revisión de los

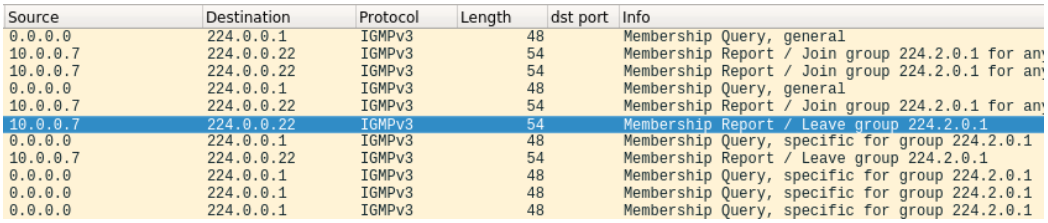
paquetes capturados con el objetivo de identificar este tipo de mensaje. correspondiente al protocolo IGMPv3.



Source	Destination	Protocol	Length	dst port	Info
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, general
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Join group 224.2.0.1
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Join group 224.2.0.1

Figura 44. *IGMPv3 Join group, S7-eth4.*
Fuente: *Elaborado por el Autor.*

Cuando un nodo receptor toma la decisión de desvincularse del grupo de multidifusión, procede a enviar un mensaje categorizado como “Leave group” a la dirección de multidifusión en específico. En este caso en particular, el host denominado “h7” con dirección IP “10.0.0.7”, abandona el grupo multicast con dirección IP “224.2.0.1”. Este procedimiento se evidencia en los paquetes que han sido capturados mediante el uso del software Wireshark.



Source	Destination	Protocol	Length	dst port	Info
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, general
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Join group 224.2.0.1 for any
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Join group 224.2.0.1 for any
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, general
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Join group 224.2.0.1 for any
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Leave group 224.2.0.1
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, specific for group 224.2.0.1
10.0.0.7	224.0.0.22	IGMPv3	54		Membership Report / Leave group 224.2.0.1
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, specific for group 224.2.0.1
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, specific for group 224.2.0.1
0.0.0.0	224.0.0.1	IGMPv3	48		Membership Query, specific for group 224.2.0.1

Figura 45. *IGMPv3 Leave group, S7-eth4.*
Fuente: *Elaborado por el Autor.*

RESULTADOS DE ESCENARIOS SDN CON PARAMETROS DE TOPOLOGÍA DE RED PREDETERMINADOS:

LATENCIA

Para llevar a cabo las mediciones de latencia, se empleó el comando ping. Este comando mide el Tiempo de Ida y Vuelta (RTT, por sus siglas en inglés), que es una medida directa de la latencia. Al utilizar las direcciones de origen y destino, se generan solicitudes del Protocolo de Mensajes de Control de Internet (ICMP). La Tabla 28 presenta las mediciones de latencia efectuadas en los escenarios de red definida por software (SDN) en diferentes topologías de red.

Tabla 28. *Latencia en escenarios con parámetros de red predeterminados.*
Fuente: *Elaborado por el Autor.*

Controlador	Topología	Tamaño [bytes]	T. min [ms]	T. max [ms]	T. avg [ms]
ODL	lineal	64	0,194	0,986	0,244
	árbol	64	0,197	1,464	0,367
	malla	64	0,219	2,010	0,412

POX	lineal	64	0,063	37,467	1,589
	árbol	64	0.081	100,627	3,382
	malla	64	0.113	101,627	2,871

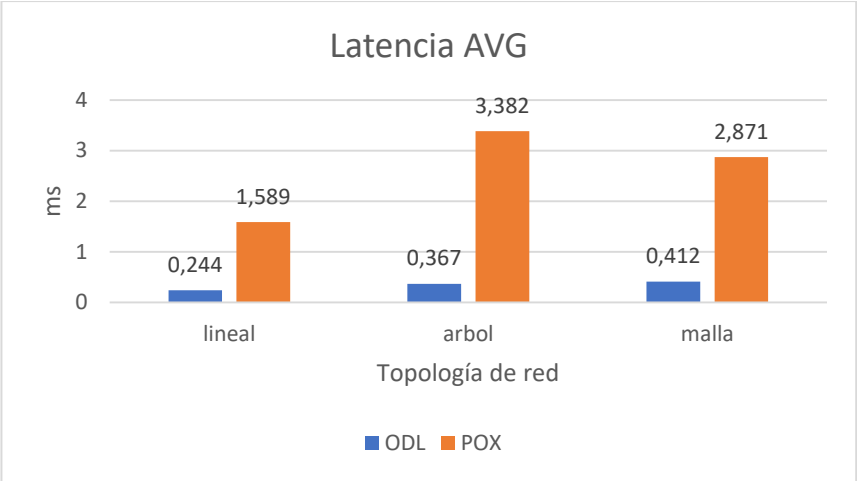


Figura 46. *Latencia promedio en escenarios SDN.*
Fuente: *Elaborado por el Autor.*

Los resultados presentan una comparación de la latencia promedio, medida en milisegundos, en tres topologías de red distintas: lineal, árbol y malla. Se utilizaron dos controladores de red diferentes para esta evaluación: ODL y POX. Según los datos obtenidos, la latencia promedio más alta se registró en la topología árbol con el controlador POX, alcanzando un valor de 3.382 ms. Por otro lado, la latencia más baja se observó en la topología lineal con el controlador ODL, con un valor de 0.244 ms. Se concluye que el controlador ODL presenta resultados de latencia bajos, por lo tanto óptimos para aplicaciones de streaming.

THROUGHPUT CON EL PROTOCOLO TCP

Para determinar el rendimiento, medido en términos de throughput con el protocolo TCP, se utilizaron los controladores de red ODL y POX en diversos escenarios de red. Para ello, se empleó la herramienta iPerf. En el servidor, se ejecutó el comando “iperf3 -s -p 5566 -i 1”, mientras que en el host de destino se utilizó el comando “iperf3 -c [ip_servidor] -b 10m -p 5566 -t 50”.

Tabla 29. *Opciones de línea de comando de la herramienta iPerf.*
Fuente: *Extraído del sitio web de iPerf.*

Parámetro	Descripción
-s	Habilita iperf en modo servidor.
-c	Habilita iperf en modo cliente.

-p	Indica el número de puerto a utilizar.
-i	Establece el intervalo de tiempo en segundos entre informes periódicos de ancho de banda.
-b	Establece el ancho de banda de destino en n bits/s (predeterminado 1 Mbit/s para UDP, ilimitado para TCP).
-t	Indica el tiempo en segundos para transmitir. El valor predeterminado es 10 segundos.
-u	Establece un flujo UDP.
-B	Permite vincular y unirse a un grupo de multidifusión. Se utiliza direcciones en el rango 224.0.0.0 a 239.255.255.255 para multidifusión.
--ttl	Time to live para multicast indica el tiempo de vida del paquete.

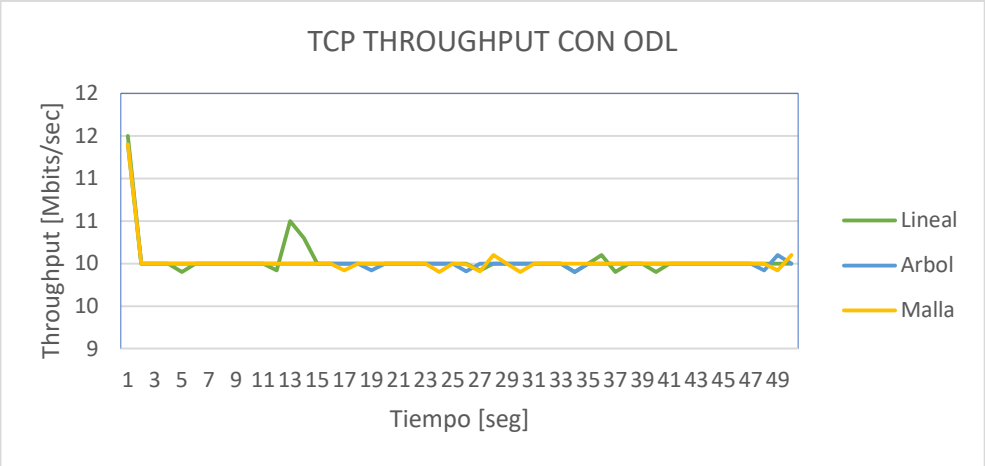


Figura 47. *Resultados de throughput mediante TCP con ODL.*
Fuente: *Elaborado por el Autor.*

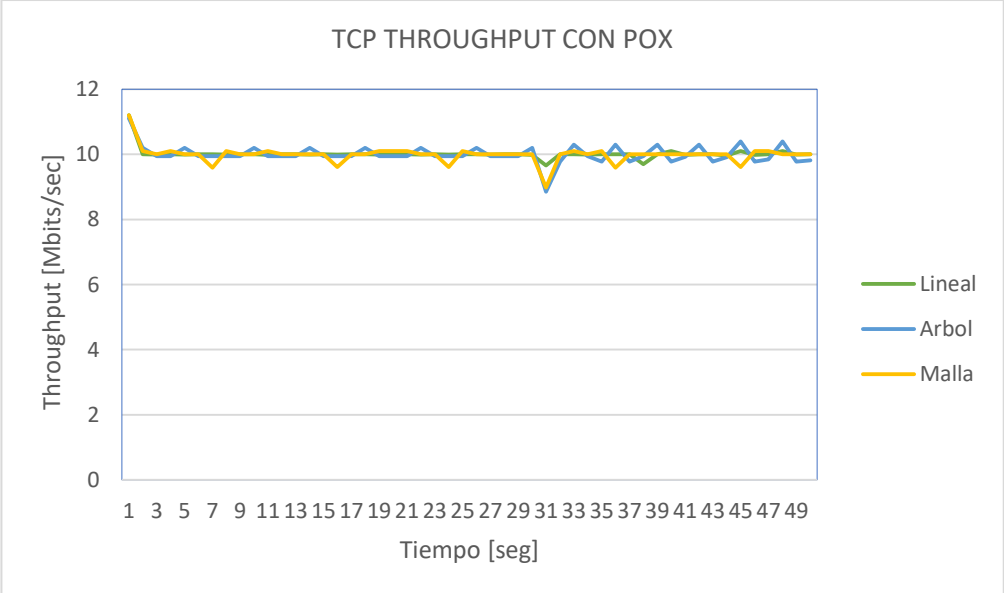


Figura 48. *Resultados de throughput mediante TCP con POX.*
Fuente: *Elaborado por el Autor.*

En las Figuras 47 y 48 se presentan los resultados obtenidos del rendimiento, medido en términos de throughput con el protocolo TCP, utilizando los controladores de red ODL y POX en escenarios de simulación con topologías de red lineal, árbol y malla. A partir de estos resultados se puede concluir que, aunque las líneas muestran ligeras fluctuaciones que indican variaciones en el rendimiento, en general, el throughput se mantiene en aproximadamente 10 Mbps para cada topología de red y controlador.

THROUGHPUT CON EL PROTOCOLO UDP

En esta sección se evaluó el rendimiento, medido en términos de throughput, con el protocolo UDP utilizando una dirección de grupo de tipo multicast. Se emplearon los controladores de red ODL y POX en diversos escenarios con las topologías de red lineal, árbol y malla. Para llevar a cabo esta evaluación, se empleó la herramienta iPerf. En el servidor, se ejecutó el comando "iperf -s -u -B 224.2.0.1 -i 1", mientras que en el host de destino se utilizó el comando "iperf -c 224.2.0.1 -u -b 10m --ttl 64 -t 50".

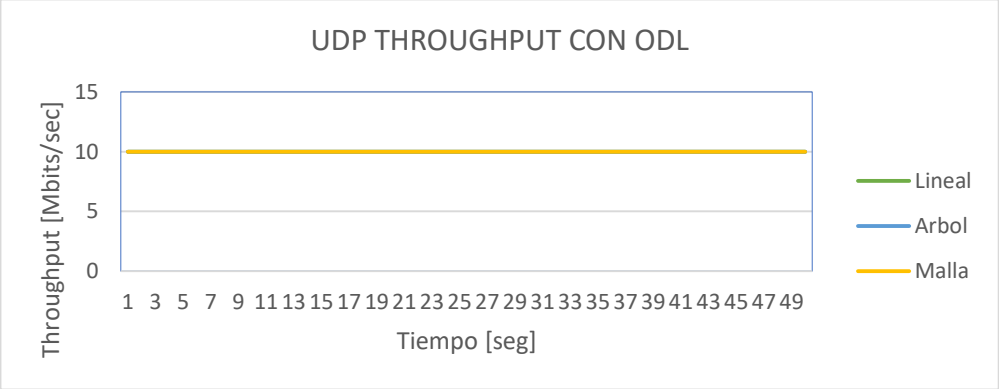


Figura 49. *Resultados de throughput mediante UDP con ODL.*
Fuente: *Elaborado por el Autor.*

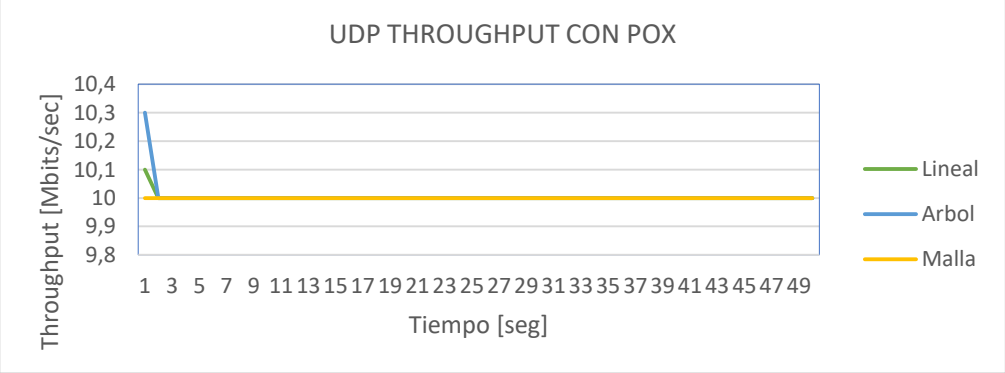


Figura 50. *Resultados de throughput mediante UDP con POX.*
Fuente: *Elaborado por el Autor.*

En las Figuras 49 y 50 se presentan los resultados obtenidos en términos de rendimiento, específicamente en lo que respecta al throughput, utilizando el protocolo UDP. Los datos analizados revelan un throughput estable de 10 Mbps. Por ende, se puede inferir que los escenarios de red SDN con los controladores ODL y POX proporcionan un rendimiento superior en términos de throughput para el protocolo UDP. Esta característica resulta particularmente atractiva para aplicaciones de multidifusión.

JITTER

En la presente sección, se llevó a cabo una evaluación del jitter en diversos escenarios de red SDN. Las topologías examinadas incluyeron las configuraciones lineal, árbol y malla. Se utilizaron los controladores de red ODL y POX, junto con una dirección de grupo multicast. Para llevar a cabo las pruebas, se empleó iPerf. El comando utilizado para el servidor fue "iperf -s -u -B 224.2.0.1 -i 1", mientras que para el host de destino se utilizó "iperf -c 224.2.0.1 -u -b 10m --ttl 64 -t 50".

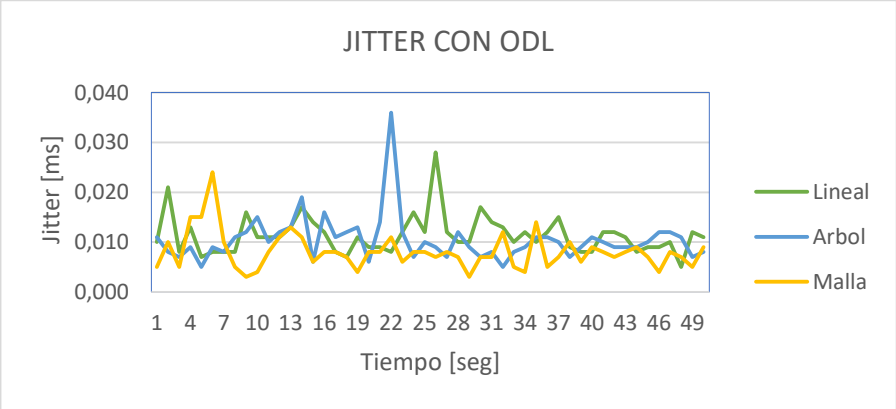


Figura 51. *Resultados de jitter con el controlador ODL.*
Fuente: *Elaborado por el Autor.*

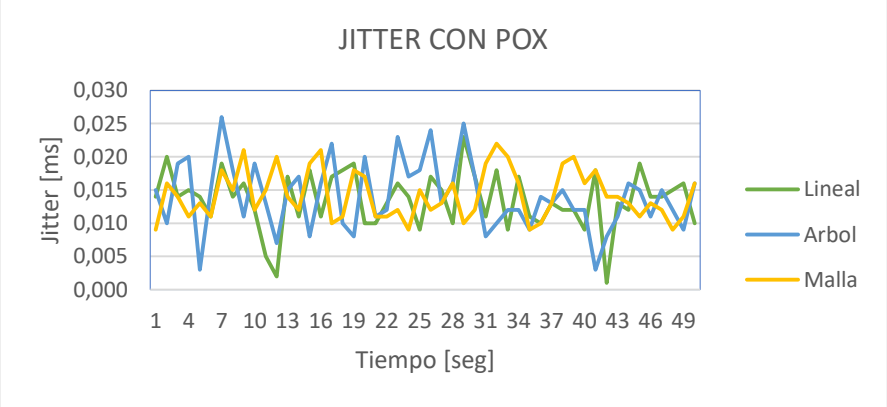


Figura 52. *Resultados de jitter con el controlador POX.*
Fuente: *Elaborado por el Autor.*

En las Figuras 51 y 52 se muestran los resultados obtenidos del jitter en los escenarios de red SDN desarrollados. Se concluye que los valores de Jitter fluctúan dependiendo de la configuración de la red y del controlador utilizado (ODL o POX). Aunque se observa que con el controlador ODL en la topología árbol se alcanza un valor máximo de jitter de 0,036ms y con el controlador POX en la topología árbol se obtiene un valor máximo de 0,026ms, estos valores se encuentran dentro de un rango aceptable para aplicaciones de transmisión de streaming multicast.

A continuación, se presentan los valores promedio obtenidos para cada parámetro de la métrica de rendimiento, según el tipo de topología de red y el controlador SDN utilizado.

Tabla 30. *Promedios de los parámetros de la métrica de rendimiento.*
Fuente: *Elaborado por el Autor.*

Controlador	Topología	Latencia AVG [ms]	Throughput TCP AVG [Mbps]	Throughput UDP AVG [Mbps]	Jitter AVG [ms]
ODL	lineal	0,244	10,037	10,000	0,011
	árbol	0,367	10,023	10,000	0,010
	malla	0,412	10,023	10,000	0,008
POX	lineal	1,589	10,013	10,002	0,014
	árbol	3,382	10,002	10,006	0,014
	malla	2,871	9,988	10,000	0,014

RESULTADOS DE ESCENARIOS SDN CON VARIACIÓN EN PARAMETROS DE LA TOPOLOGÍA DE RED:

LATENCIA Y PERDIDA DE PAQUETES

Se llevaron a cabo mediciones de latencia y pérdida de paquetes en los escenarios de red SDN, variando los parámetros de la topología de red de acuerdo a las especificaciones de la Tabla 10. Para la evaluación, se utilizó el comando ‘ping’. La Tabla 30 muestra las mediciones realizadas en los escenarios de Red Definida por Software (SDN) con diferentes topologías de red.

Tabla 31. *Mediciones de latencia en escenarios de red SDN.*
Fuente: *Elaborado por el Autor.*

Controlador	Topología	Tamaño [bytes]	T. min [ms]	T. max [ms]	T. avg [ms]	Packet Loss
ODL	lineal	64	30,293	31,442	30,465	22%
	árbol	64	24,234	51,475	25,081	12%
	malla	64	46,396	96,900	47,727	12%
POX	lineal	64	30,335	148,705	35,583	38%
	árbol	64	24,348	117,699	27,934	10%
	malla	64	36,261	212,909	41,416	10%

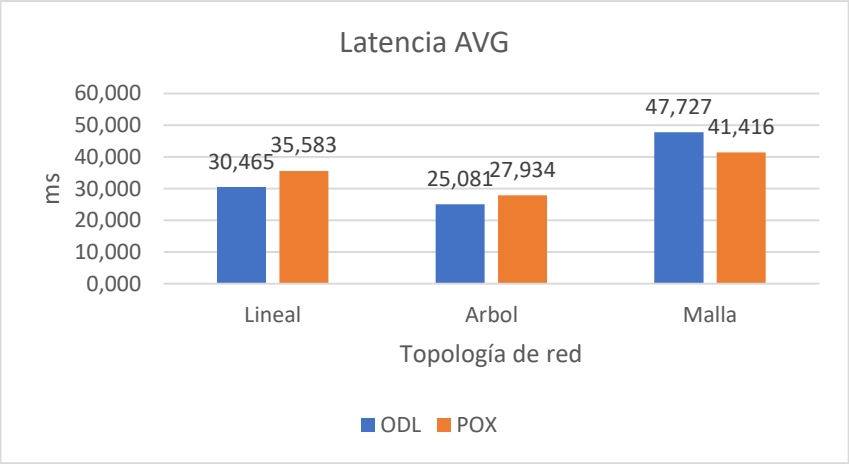


Figura 53. *Latencia promedio en escenarios de red SDN.*
Fuente: *Elaborado por el Autor.*

Los resultados obtenidos en la Figura 53 indican que la latencia promedio más alta se registró en la topología de malla con el controlador ODL, alcanzando un valor de 47,73 ms. En comparación con la latencia promedio más baja que se observó en la topología árbol con el controlador ODL, donde se registró un valor de 25,08 ms.

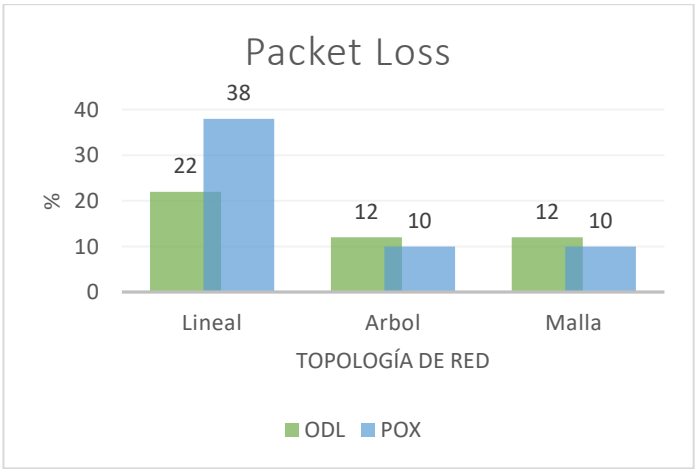


Figura 54. *Perdida de paquetes en escenarios de red SDN.*
Fuente: *Elaborado por el Autor.*

En la Figura 54 se observan los resultados obtenidos de perdida de paquetes en los escenarios de red SDN. Se obtuvo el valor más alto de 38% en un escenario de topología de red lineal con el controlador POX y el valor más bajo de 10% en las topologías de red malla y árbol con POX. Los resultados obtenidos demuestran cómo el incremento de la latencia y la pérdida de paquetes afectan y deterioran el rendimiento de una red SDN.

THROUGHPUT CON EL PROTOCOLO TCP

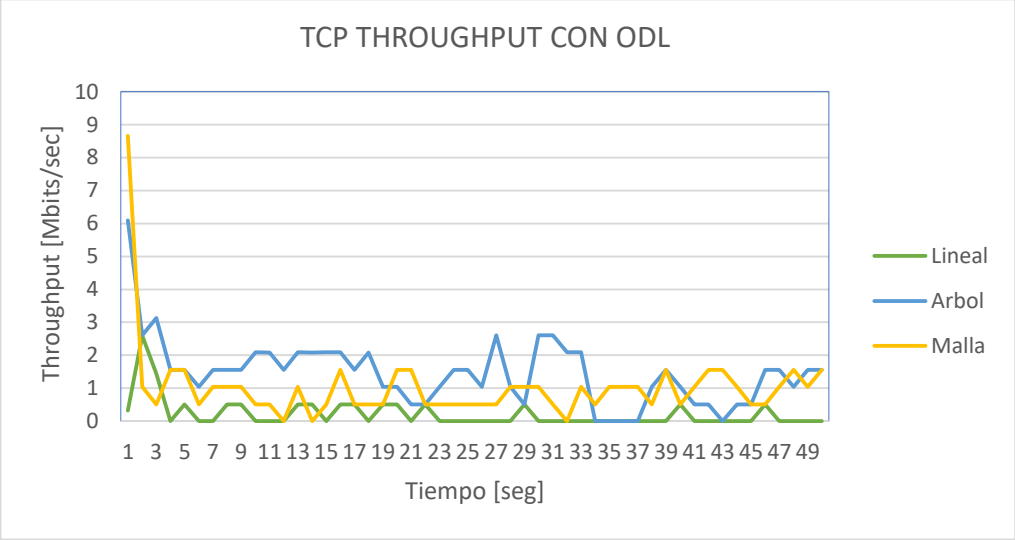


Figura 55. *Resultados de throughput mediante TCP con ODL.*
Fuente: *Elaborado por el Autor.*

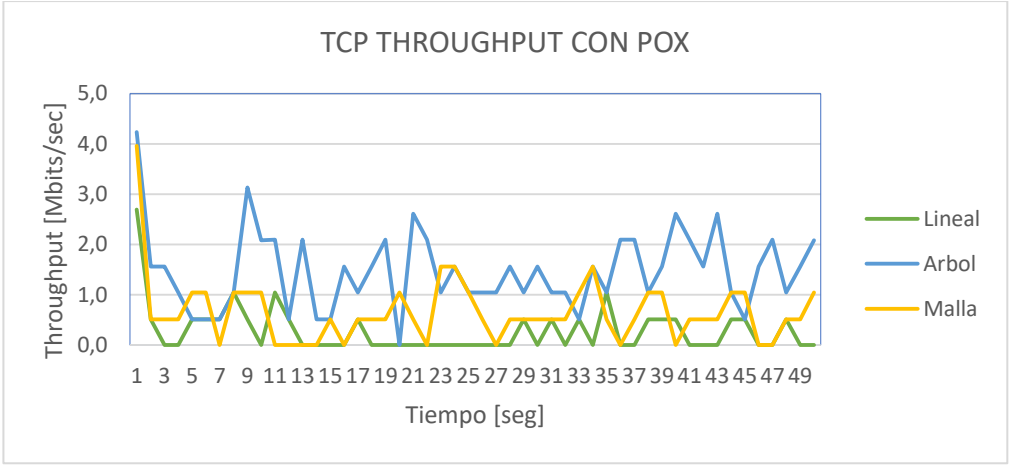


Figura 56. *Resultados de throughput mediante TCP con POX.*
Fuente: *Elaborado por el Autor.*

En las Figuras 55 y 56 se muestran los resultados obtenidos en términos de rendimiento, específicamente el throughput para los escenarios de red SDN. A partir de estos resultados, se puede concluir que la introducción de variaciones en los parámetros del enlace de la topología de red conduce a una disminución

significativa en el rendimiento de la red. En particular, la topología malla muestra un valor máximo de throughput aproximado de solo 8,66 Mbps con ODL y 4,23 Mbps en una topología árbol con POX. Por otro lado, las demás topologías de red muestran resultados aún más bajos, llegando a 0 Mbps en su punto más bajo.

THROUGHPUT CON EL PROTOCOLO UDP

En esta sección, se llevó a cabo una evaluación del rendimiento de los escenarios de red SDN, específicamente en términos de throughput, utilizando el protocolo UDP y la dirección de grupo multicast "224.2.0.1" con los controladores de red ODL y POX. Este análisis implicó la variación de los parámetros de la topología de la red, siguiendo las especificaciones detalladas en la Tabla 10.

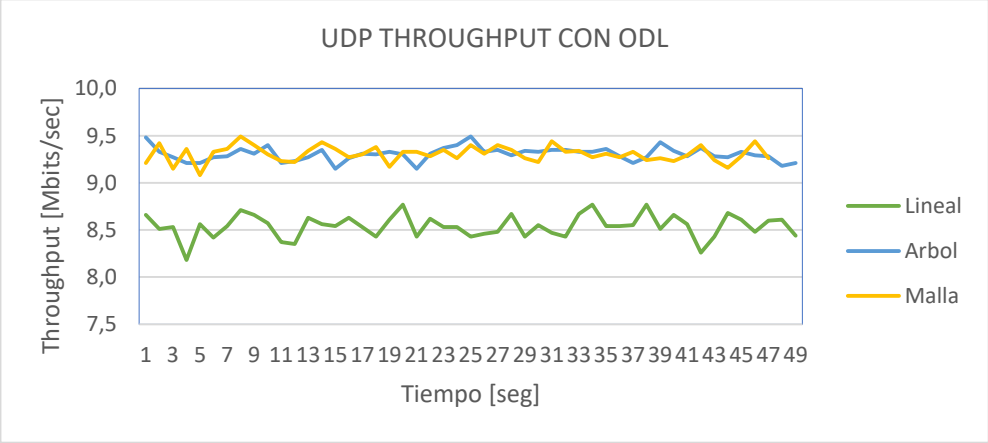


Figura 57. *Resultados de throughput mediante UDP con ODL.*
Fuente: *Elaborado por el Autor.*

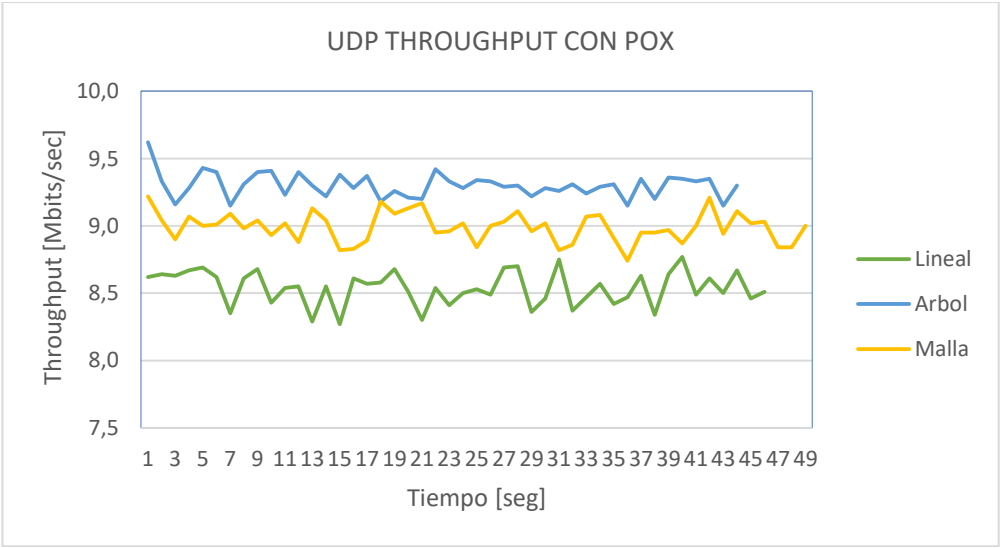


Figura 58. *Resultados de throughput mediante UDP con POX.*
Fuente: *Elaborado por el Autor.*

En las Figuras 57 y 58 se presentan los resultados de rendimiento, cuantificados en términos de throughput, correspondientes a los escenarios de simulación implementados. A partir de estos datos, se puede inferir que todas las topologías de red, empleando los controladores de red ODL y POX, experimentan una disminución en el rendimiento, medido en términos de throughput, conforme se incrementa el tiempo. Esta disminución afecta de manera notable a los escenarios con topología de red lineal que utilizan ODL y POX, alcanzando un valor promedio de throughput de 8,5 Mbps, mientras que las demás topologías no superan los 9,6 Mbps. Esta observación sugiere que la eficiencia de la red podría verse comprometida al introducir variaciones en los parámetros del enlace de la topología de red.

JITTER

En esta sección, se realizó un análisis del Jitter en los escenarios de red SDN, empleando el protocolo UDP y la dirección de grupo de tipo multicast “224.2.0.1” con los controladores de red ODL y POX. Este estudio implicó la modificación de los parámetros de la topología de la red, de acuerdo con las especificaciones presentadas en la Tabla 10.

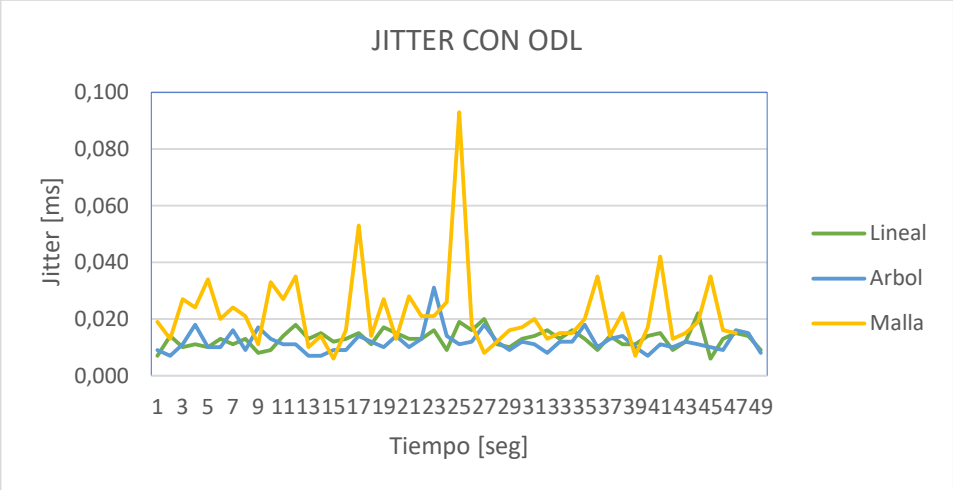


Figura 59. *Resultados de jitter con el controlador ODL.*
Fuente: *Elaborado por el Autor.*

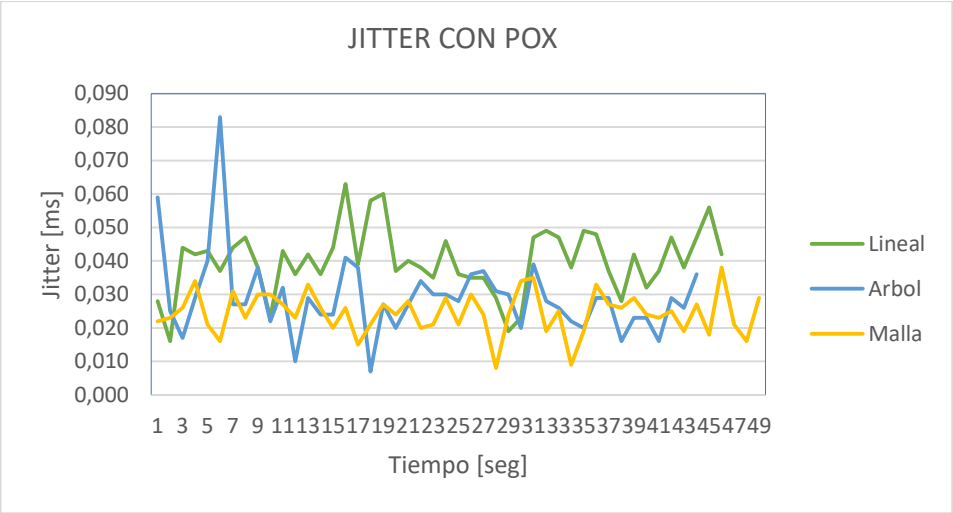


Figura 60. *Resultados de jitter con el controlador POX.*
Fuente: *Elaborado por el Autor.*

Las Figuras 59 y 60 muestran los resultados obtenidos del jitter en los escenarios de red SDN desarrollados. A partir de estos resultados, se puede concluir que los valores de jitter fluctúan dependiendo de la configuración de la red y del controlador utilizado (ODL o POX). Se observa que al introducir variaciones en los parámetros del enlace de la topología de red, se produce un aumento considerable del jitter con un valor máximo de 0,093ms en una topología malla con ODL y 0,083ms en una topología árbol en POX. Este aumento puede ocasionar interrupciones y una disminución de la calidad del audio y video en aplicaciones de streaming de tipo multicast.

A continuación, se presentan los valores promedio obtenidos para cada parámetro de la métrica de rendimiento, según el tipo de topología de red y el controlador SDN utilizado.

Tabla 32. *Promedios de los parámetros de la métrica de rendimiento.*
Fuente: *Elaborado por el Autor.*

Controlador	Topología	Latencia AVG [ms]	Perdida de Paquetes [%]	Throughput TCP AVG [Mbps]	Throughput UDP AVG [Mbps]	Jitter AVG [ms]
ODL	lineal	30,465	22	0,220	8,541	0,013
	árbol	25,081	12	1,464	9,305	0,012
	malla	47,727	12	1,022	9,307	0,022
POX	lineal	35,583	38	0,279	8,537	0,040
	árbol	27,934	10	1,468	9,308	0,030
	malla	41,416	10	0,657	8,991	0,025

TRANSMISIÓN DE STREAMING MULTICAST EN ESCENARIO CON VARIACIÓN EN PARÁMETROS DE TOPOLOGÍA DE RED

Para evaluar cómo una transmisión de streaming multicast se ve afectada en escenarios con variación en los parámetros del enlace de la topología de red, se presenta en la Figura 61 la implementación de streaming en el escenario de topología de red tipo malla con el controlador POX utilizando el protocolo RTP y el grupo de multidifusión “224.2.0.1”. Este análisis permite observar cómo las variaciones en los parámetros del enlace pueden influir en la calidad de la transmisión de video en streaming.

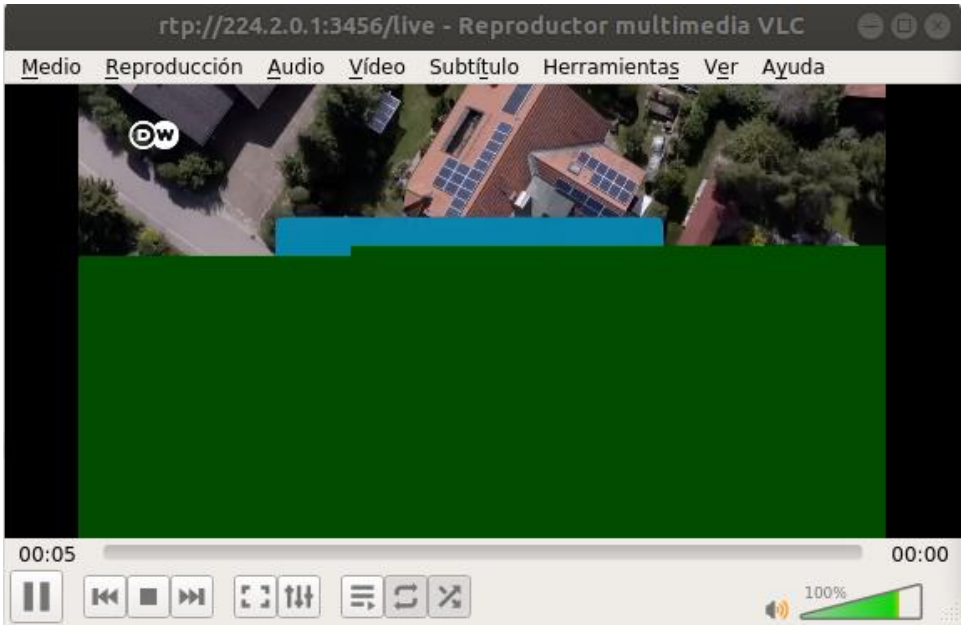


Figura 61. *Visualización de video en streaming en escenario con adición de tasa de pérdida y tiempo de retardo.*
Fuente: *Elaborado por el Autor.*

En resumen, los indicadores de rendimiento de la red, que se derivan de los entornos de Red Definida por Software (SDN), son factores determinantes que pueden influir de manera significativa en la calidad de una transmisión de streaming en multidifusión. Tal como se evidencia en la Figura 61, al introducir variaciones en los parámetros de los enlaces de la topología de red, se encontró que una pérdida de paquetes superior al 5%, combinada con un tiempo de retardo de hasta 10 ms, generaba fluctuaciones significativas en la calidad del video y audio, ocasionando retrasos e interrupciones y afectando la experiencia del usuario final.

CONCLUSIONES

- Se demostró que la arquitectura de Redes Definidas por Software (SDN), mediante el uso de los protocolos OpenFlow e IGMPv3, es una solución eficaz para la gestión centralizada y eficiente del tráfico de red. Esta tecnología no solo optimiza la transmisión de servicios multimedia al reducir la latencia y mejorar la calidad del servicio, sino que también facilita la entrega eficiente de contenido en streaming a múltiples usuarios.

- Las simulaciones realizadas abarcaron la creación de diversas topologías de red mediante el uso de los controladores SDN ODL y POX. Este enfoque permitió experimentar con diferentes configuraciones y protocolos. Como resultado, se consiguió que un grupo multicast tuviera la capacidad de conectar simultáneamente hasta nueve dispositivos sin afectar la calidad de la transmisión en streaming.

- La evaluación de los resultados obtenidos a partir de las métricas y parámetros de red evidenció que un aumento en el retardo y la pérdida de paquetes impacta negativamente en la calidad del servicio de video en streaming. En los escenarios analizados, se encontró que una pérdida de paquetes superior al 5%, combinada con un tiempo de retardo de hasta 10 ms, generaba fluctuaciones significativas en la calidad del video y audio, ocasionando retrasos e interrupciones y afectando la experiencia del usuario final.

- La evaluación de escenarios SDN con adición de tasa de pérdida y retardo indicó que el controlador ODL con topología de árbol proporcionó el mejor rendimiento general. Esta combinación presentó la menor latencia promedio (25.081 ms) y una pérdida de paquetes relativamente baja (12%). Además, alcanzó altos promedios de throughput tanto para TCP (1.464 Mbps) como para UDP (9.305 Mbps), con un jitter mínimo de 0.012 ms. Estos resultados sugieren que para aplicaciones de streaming que requieren baja latencia y alto throughput, la combinación de ODL y topología de árbol es la más adecuada.

RECOMENDACIONES

- Para garantizar una transmisión eficiente y exitosa de streaming mediante multidifusión, es esencial considerar que esto solo es viable a través del protocolo UDP, Esto se debe a que el protocolo TCP no es adecuado para multidifusión debido a sus características de conexión y retransmisión.
- Dada la importancia del protocolo IGMPv3 en la gestión de grupos multicast, se recomienda explorar e implementar protocolos avanzados que puedan mejorar aún más la eficiencia del servicio de streaming, lo cual es crucial para la calidad de la experiencia del usuario final.
- Establecer un sistema de monitoreo continuo sobre el controlador SDN que evalúe el rendimiento de la red en tiempo real. Esto no solo permitirá identificar problemas de forma temprana, sino también ajustar los recursos y optimizar el desempeño del servicio de streaming en situaciones de carga variable.

BIBLIOGRAFÍA

[1] A. Canovas, A. Rego, O. Romero y J. Lloret, «A robust multimedia traffic SDN-Based management system using patterns and models of QoE estimation with BRNN,» 2020. [En línea]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1084804519303583?via%3Dihub. [Último acceso: 25 06 2022].

[2] Networking y C. Visual, «VNI Complete Forecast Highlights,» 1 10 2018. [En línea]. Available: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2022_Forecast_Highlights.pdf. [Último acceso: 17 03 2022].

[3] E. Liotou, K. Samdanis, E. Pateromichelakis y N. Passas, «QoE-SDN APP: A Rate-guided QoE-aware SDN-APP for HTTP Adaptive Video Streaming,» *IEEE Journal on Selected Areas in Communications,* vol. 36, nº doi: 10.1109/JSAC.2018.2815421, pp. 598 - 615, March 2018.

[4] López y S. Burgos, «Tráfico multimedia con Mininet en redes,» 4 07 2018. [En línea]. Available: http://hdl.handle.net/10251/109341. [Último acceso: 1 07 2022].

[5] I. Red Hat, «Virtualización de red,» Red Hat, 23 Marzo 2021. [En línea]. Available: https://www.redhat.com/en/topics/virtualization/what-is-network-virtualization. [Último acceso: 27 Marzo 2022].

[6] E. Cuesta, E. Paola, M. Quintero, J. Carlos, C. Torres y S. Michael, «Análisis del protocolo Openflow usando mininet para una red SDN Openflow,» 14 10 2021. [En línea]. Available: https://hemeroteca.unad.edu.co/index.php/wpecbti/article/view/4811/5183. [Último acceso: 2022 04 22].

[7] C. Koch, S. Hacker y D. Hausheer, «VoDCast: Efficient SDN-based multicast for video on demand,» *IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks*, nº doi: 10.1109/WoWMoM.2017.7974319, pp. 1-6, 2017.

[8] T. Grajales, «Tipos de investigación,» 27 03 2000. [En línea]. Available: https://cmapspublic2.ihmc.us/rid=1RM1F0L42-VZ46F4-319H/871.pdf. [Último acceso: 2 05 2022].

[9] M. Li, Z. Mao y J. Rexford, «Toward Software-Defined Campus Networks,» *ACM SIGCOMM Computer Communication Review,* nº 42(4), pp. 84-89, 2012.

[10] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg, S. Azodolmolky y S. Uhlig, «Software-Defined Networking: A Comprehensive Survey,» *Proceedings of the IEEE,* nº 103(1), pp. 14-76, 2015.

[11] S. Nanda y T.-c. Chiueh, «A Survey on Virtualization Technologies,» *Department of Computer Science SUNY at Stony Brook,* nº 11794-4400, pp. 5-13, 2018.

[12] A. Guo, «Network virtualization,» *IEEE in Optical Fiber Communication Conference,* pp. 1-3, 2014.

[13] N. M, K. Chowdhury y R. Boutaba, «Network virtualization: state of the art and research challenges,» *IEEE Commun Mag,* vol. 47, nº 7, pp. 20-26, 2009.

[14] C. Tipantuña y P. Yanchapaxi, «Network functions virtualization: An overview and open-source projects," 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM),» de *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, Salinas, Ecuador, doi: 10.1109/ETCM.2017.8247541, 2017, pp. 1-6.

[15] K. Kaur, V. Mangat y K. Kumar, «Architectural framework, research issues and challenges of network function virtualization,» *8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO),* p. 474–478, 2020.

[16] C. Calcaterra, A. Carmenini, A. Marotta, U. Bucci y D. Cassioli, «MaxHadoop: An efficient scalable emulation tool to test SDN protocols in emulated Hadoop environments,» *J. Netw. Syst. Manag,* vol. 28, nº 4, p. 1610–1638, 2020.

[17] Bawman.com, «Software Defined Networking,» Mayo 2013. [En línea]. Available: https://www.bawman.com/BAWMAN/downloadables/BAWMAN-CONET-Presentation-2013-05.pdf. [Último acceso: 20 Abril 2023].

[18] «Chapter 1: Introduction — Software-Defined Networks: A Systems Approach Version 2.1-dev documentation,» Systemsapproach.org, [En línea]. Available: https://sdn.systemsapproach.org/intro.html. [Último acceso: 22 Abril 2023].

[19] O. N. Foundation, «SDN architecture,» opennetworking.org, Junio 2014. [En línea]. Available: https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf. [Último acceso: 24 Abril 2023].

[20] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh y M. Keshtgary, «A survey on SDN, the future of networking,» *J. Adv. Comput. Sci. Technol,* vol. 3, nº 2, p. 232, 2014.

[21] S. Denazis, H. Salim, J. Meyer y O. Koufopavlou, «Software-Defined Networking (SDN): Layers and Architecture Terminology,» RFC Editor, January 2015. [En línea]. Available: https://www.rfc-editor.org/rfc/rfc7426.html.

[22] D. B. Rawat y S. R. Reddy, «Software defined networking architecture, security and energy efficiency: A survey,» *IEEE Commun. Surv. Tutor,* vol. 19, nº 1, p. 325–346, 2017.

[23] D. Hoang y M. Pham, «On software-defined networking and the design of SDN controllers,» de *2015 6th International Conference on the Network of the Future (NOF),* Montreal, QC, Canada, doi: 10.1109/NOF.2015.7333307, 2015, pp. 1-3.

[24] O. Salman, I. Elhajj, A. Kayssi y A. Chehab, «SDN controllers: A comparative study,» *18th Mediterranean Electrotechnical Conference (MELECON),* pp. 1-6, 2016.

[25] L. Zhu, M. Karim, K. Sharif, F. Li, X. Du y M. Guizani, «SDN Controllers: Benchmarking & Performance Evaluation,» *arXiv [cs.NI],* p. 14, 2019.

[26] Hewlett-Packard Development Company, «VAN SDN Controller Programming Guide,» 2013, pp. 5-6.

[27] G. Mocha y J. Celleri, «Análisis Comparativo de Protocolos de Comunicación para Redes definidas por software,» *Hamut´ay,* vol. 7, nº 3, http://dx.doi.org/10.21503/hamu.v7i3.2190, pp. 39-50, 2020.

[28] «Sdxcentral.com,» [En línea]. Available: http://sdxcentral.com/networking/sdn/definitions/what-the-definition-of-softwaredefined-networking-sdn/what-is-openflow/. [Último acceso: 2 febrero 2023].

[29] C. Valdivieso, B. Peral, B. López y V. García, «SDN: Evolution and opportunities in the development IoT applications,» *Int. J. Distrib. Sens. Netw.,* vol. 10, nº 5, p. 735142, 2014.

[30] B. Isyaku, M. Zahid, B. Kamat, A. Bakar y F. Ghaleb, «Software defined networking flow table management of OpenFlow switches performance and security challenges: A survey,» *Future Internet,* vol. 12, nº 9, p. 147, 2020.

[31] openflow.org, «OpenFlow Switch Specification, 1.0.0,» December 2009. [En línea]. Available: http://archive.openflow.org/documents/ openflow-spec-v1.0.0.pdf. [Último acceso: 5 marzo 2023].

[32] openflow.org, «OpenFlow Switch Specification, 1.1.0,» february 2011. [En línea]. Available: http://archive.openflow.org/documents/ openflow-spec-v1.1.0.pdf. [Último acceso: 5 marzo 2023].

[33] opennetworking.org, «OpenFlow Switch Specification, 1.2.0,» December 2011. [En línea]. Available: https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onfspecifications/openflow/ openflow-spec-v1.2.pdf. [Último acceso: 5 marzo 2023].

[34] opennetworking.org, «OpenFlow Switch Specification, 1.3.0,» june 2012. [En línea]. Available: https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onfspecifications/openflow/ openflow-spec-v1.3.0.pdf. [Último acceso: 5 marzo 2023].

[35] opennetworking.org, «OpenFlow Switch Specification, 1.4.0,» October 2013. [En línea]. Available: https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onfspecifications/openflow/ openflow-spec-v1.4.0.pdf. [Último acceso: 6 marzo 2023].

[36] opennetworking.org, «OpenFlow Switch Specification, 1.5.0,» 2014. [En línea]. Available: https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onfspecifications/openflow/ openflow-switch-v1.5.0.noipr.pdf. [Último acceso: 7 marzo 2023].

[37] Edu.tw. [En línea]. Available: https://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf. [Último acceso: 8 marzo 2023].

[38] E. Rojas y e. al, «Are we ready to drive Software-defined networks? A comprehensive survey on management tools and techniques,» *ACM Comput. Surv,* vol. 51, nº 2, pp. 1-35, 2019.

[39] A. Gillis, «What is video streaming? Definition, meaning and how it works,» Unified Communications, 12 Nov 2021. [En línea]. Available: https://www.techtarget.com/searchunifiedcommunications/definition/streaming-video. [Último acceso: 12 marzo 2023].

[40] D. Madhuri y P. Reddy, «Performance comparison of TCP, UDP and SCTP in a wired network,» *in 2016 International Conference on Communication and Electronics Systems (ICCES),* 2016.

[41] L. Rosencrance, G. Lawton y C. Moozakis, «User Datagram Protocol (UDP),» Networking, 04 Oct 2021. [En línea]. Available: https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol. [Último acceso: 4 abril 2023].

[42] C. Pakanati, M. Padmavathamma y N. Reddy, «Performance comparison of TCP, UDP, and TFRC in wired networks,» *in 2015 IEEE International Conference on Computational Intelligence & Communication Technology,* pp. 1-7, 2015.

[43] H. Schulzrinne, A. Rao y R. Lanphier, «Real Time Streaming Protocol (RTSP),» RFC Editor, April 1998. [En línea]. Available: https://www.rfc-editor.org/rfc/rfc2326. [Último acceso: 15 Abril 2023].

[44] Adobe.com. [En línea]. Available: https://wwwimages2.adobe.com/content/dam/cc/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf. [Último acceso: 25 abril 2023].

[45] R. Frederick, «Network working group audio-video transport working group request for comments: 1889 H. schulzrinne category: Standards track GMD fokus S. casner precept software, inc,» Rfc-editor.org, January 1996. [En línea]. Available: https://www.rfc-editor.org/pdfrfc/rfc1889.txt.pdf. [Último acceso: 7 mayo 2023].

[46] S. Krishnan, «Network Protocols Handbook,» Saratoga CA 95070 USA, Javvin Technologies Inc, 2014, pp. 144-145.

[47] G. Fairhurst, «Unicast,» 10 Marzo 2020. [En línea]. Available: https://erg.abdn.ac.uk/users/gorry/course/intro-pages/uni-b-mcast.html. [Último acceso: 12 mayo 2023].

[48] S. Kaur, K. Singh y S. Y, «A comparative analysis of unicast, multicast, broadcast and anycast addressing schemes routing in MANETs,» International Journal of Computer Applications, 2016. [En línea]. Available: https://doi.org/10.5120/ijca2016908001.

[49] A. Caputo, «Digital Video Surveillance and Security,» Burlington, MA 01803, USA, Elsevier, 2014, pp. 112-114.

[50] V. Damjanovski, «CCTV Networking and Digital Technology,» Butterworth-Heinemann Elsevier, 2014. [En línea]. Available: https://doi.org/10.1016/C2010-0-68632-9. [Último acceso: 2 junio 2023].

[51] Universidad Autónoma del Estado de Hidalgo, «Comunicaciones multicast,» Edu.Mx, [En línea]. Available: https://www.uaeh.edu.mx/scige/boletin/huejutla/n9/r1.html. [Último acceso: 5 Junio 2023].

[52] D. Medhi y K. Ramasamy, «Multicast Routing. Network Routing,» Elsevier, 2018. [En línea]. Available: doi:10.1016/b978-0-12-800737-2.00010-7 . [Último acceso: 12 junio 2023].

[53] S. Deering, «RFC 1112: Host extensions for IP multicasting.,» IETF Datatracker Stanford University, 1 August 1989. [En línea]. Available:

https://datatracker.ietf.org/doc/html/rfc1112. [Último acceso: 15 Junio 2023].

[54] W. Fenner, «RFC 2236: Internet group management protocol, version 2,» IETF Datatracker Xerox PARC, 1 November 1997. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc2236. [Último acceso: 17 Junio 2023].

[55] B. Cain, S. Deering, B. Kouvelas y A. Thyagarajan, «RFC 3376: Internet Group Management Protocol, Version 3,» IETF Datatracker, Ericsson, 17 October 2022. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc3376. [Último acceso: 20 Junio 2023].

[56] A. Garcés y V. Directora, «Estudio sobre las Redes Definidas por Software (SDN) para mejorar la aplicación del Streaming de video en las Redes de Distribución de Contenidos (CDN) en el Ecuador,» ESPOL. FIEC, 2020. [En línea]. Available: https://www.dspace.espol.edu.ec/handle/123456789/56377. [Último acceso: 25 Junio 2023].

[57] V. Echeverria y C. Jesús, «Diseño y simulacion de un prototipo de red definida por software (SDN) usando el protocolo OpenFlow,» Universidad de Guayaquil Facultad de Ciencias Matemáticas y Físicas Carrera de Ingeniería en Networking y Telecomunicaciones, Diciembre 2015. [En línea]. Available: http://repositorio.ug.edu.ec/handle/redug/11996. [Último acceso: 3 Julio 2023].

[58] L. Barlocco y B. Pugliesi, «Sistema de distribución Multicast mediante redes definidas por software (Proyecto),» Universidad ORT Uruguay, Facultad de Ingeniería, 2019. [En línea]. Available: https://dspace.ort.edu.uy/handle/ vol. 20, p. 500. [Último acceso: 7 Julio 2023].

[59] Python.org, «PYTHON,» Python Software Foundation, [En línea]. Available: https://www.python.org/about/. [Último acceso: 2 Agosto 2023].

[60] The Linux Foundation, «OpenDaylight,» OpenDaylight Project The Linux Foundation, 2023. [En línea]. Available: https://www.opendaylight.org/about/platform-overview. [Último acceso: 5 Agosto 2023].

[61] Rovarga, «Mirror of the OpenDaylight dlux gerrit project,» Github OpenDaylight DLUX, 29 Septiembre 2020. [En línea]. Available: https://github.com/opendaylight/dlux.

[62] «Java,» Oracle.com, [En línea]. Available: https://www.oracle.com/java/. [Último acceso: 9 Julio 2023].

[63] MurphyMc, «N. O. X. Repo, pox: The POX network software platform,» github.com, 20 Mayo 2020. [En línea]. Available: https://github.com/noxrepo/pox. [Último acceso: 25 Julio 2023].

[64] A. Craig, «GroupFlow: Multicast routing in software defined networks,» Github.io, 2014. [En línea]. Available: http://alexcraig.github.io/GroupFlow/. [Último acceso: 26 Julio 2023].

[65] «Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet,» Mininet Project Contributors, 2022. [En línea]. Available: http://mininet.org/. [Último acceso: 27 Julio 2023].

[66] «MININET,» Open Networking Foundation, 2023. [En línea]. Available: https://opennetworking.org/mininet/. [Último acceso: 28 Julio 2023].

[67] The Linux Foundation, «Open vSwitch,» Openvswitch.org, 2016. [En línea]. Available: http://openvswitch.org. [Último acceso: 28 Julio 2023].

[68] B. Linkletter, «Open Source Routing and Network Simulation blog,» brianlinkletter, 2 April 2015. [En línea]. Available: https://www.brianlinkletter.com/2015/04/how-to-use-miniedit-mininets-graphical-user-interface/. [Último acceso: 29 Julio 2023].

[69] VideoLAN Organization, «Official download of VLC media player, the best Open Source player - VideoLAN,» Videolan.org, [En línea]. Available: https://www.videolan.org/vlc/. [Último acceso: 30 Julio 2023].

[70] «About FFmpeg,» Ffmpeg.org, [En línea]. Available: https://ffmpeg.org/about.html. [Último acceso: 30 Julio 2023].

[71] Aler9, «MediaMTX,» bluenviron, [En línea]. Available: https://github.com/bluenviron/mediamtx. [Último acceso: 30 Julio 2023].

[72] S. Richard, E. Warnicke y U. Lamping, «Wireshark User's Guide Version 4.1.1,» Wireshark, [En línea]. Available: https://www.wireshark.org/download/docs/Wireshark%20User's%20Guide.pdf. [Último acceso: 31 Julio 2023].

[73] Gueant, V. (s/f), «iPerf,» iPerf - The TCP, UDP and SCTP network bandwidth measurement tool., [En línea]. Available: https://iperf.fr/. [Último acceso: 18 10 2023].

[74] OpenDaylight Project, «OpenDaylight user interface (DLUX) — OpenDaylight documentation nitrogen documentation,» OpenDaylight Project Readthedocs.io, 2016. [En línea]. Available: https://test-odl-

docs.readthedocs.io/en/latest/getting-started-guide/common features/dlux.html. [Último acceso: 10 Agosto 2023].

[75] E. McCauley, «Installing POX — POX Manual Current documentation,» Github.io, 2015. [En línea]. Available: https://noxrepo.github.io/pox-doc/html/. [Último acceso: 12 Agosto 2023].

[76] A. Craig, «Module API and Detailed Documentation,» GroupFlow, 2014. [En línea]. Available: http://alexcraig.github.io/GroupFlow/module_api.html. [Último acceso: 14 Agosto 2023].

ANEXOS

Anexo 1. *Script que incorpora una topología de red lineal con ODL.*
Fuente: *Elaborado por el Autor.*

```
#!/usr/bin/env python
# coding=utf-8
from mininet.net import Mininet
from mininet.topo import *
from mininet.net import *
from mininet.node import OVSSwitch, UserSwitch
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import Node, RemoteController
from time import sleep, time
from datetime import datetime
from subprocess import call


class SDN_Topo( Topo ):
    def __init__( self ):
        # Inicializar topología
        Topo.__init__( self )

        # Añadir hosts y switches
        h1 = self.addHost('h1', ip='10.0.0.1')
        h2 = self.addHost('h2', ip='10.0.0.2')
        h3 = self.addHost('h3', ip='10.0.0.3')
        h4 = self.addHost('h4', ip='10.0.0.4')
        h5 = self.addHost('h5', ip='10.0.0.5')
        h6 = self.addHost('h6', ip='10.0.0.6')
        h7 = self.addHost('h7', ip='10.0.0.7')
        h8 = self.addHost('h8', ip='10.0.0.8')
        h9 = self.addHost('h9', ip='10.0.0.9')
        serv1 = self.addHost('serv1', ip='10.1.1.1')
        serv2 = self.addHost('serv2', ip='10.1.1.2')
        serv3 = self.addHost('serv3', ip='10.1.1.3')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')


        # Añadir enlaces (links)
        self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
        self.addLink(s2, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s1, h1, bw = 1000, use_htb = True)
```

```
        self.addLink(s1, h2, bw = 1000, use_htb = True)
        self.addLink(s1, h3, bw = 1000, use_htb = True)
        self.addLink(s2, h4, bw = 1000, use_htb = True)
        self.addLink(s2, h5, bw = 1000, use_htb = True)
        self.addLink(s2, h6, bw = 1000, use_htb = True)
        self.addLink(s3, h7, bw = 1000, use_htb = True)
        self.addLink(s3, h8, bw = 1000, use_htb = True)
        self.addLink(s3, h9, bw = 1000, use_htb = True)
        self.addLink(serv1, s1, bw = 1000, use_htb = True)
        self.addLink(serv2, s2, bw = 1000, use_htb = True)
        self.addLink(serv3, s3, bw = 1000, use_htb = True)


    def get_host_list(self):
        return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
'serv3']

    def get_switch_list(self):
        return ['s1', 's2', 's3']


def SDN_Config(topo, hosts = [], interactive = False):

    # Configuración de red para usar el controlador externo ODL.
    net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
link=TCLink, build=False, autoSetMacs=True)
    net.addController('odl', RemoteController, ip = '127.0.0.1', port = 6633)
    net.start()

    # Configurar interface de control en switches de red
    for switch_name in topo.get_switch_list():
        net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
        net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
        net.get('odl').cmd('route add -host ' + net.get(switch_name).IP() + '
dev lo')

    # Configuración de hosts para soporte multicast con igmpv3
    for host_name in topo.get_host_list():
        # Configurar rutas multicast para interfaces virtuales
        net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
eth0')
        # Configurar igmpv3 en cada host
        net.get(host_name).cmd('echo 3 >
/proc/sys/net/ipv4/conf/all/force_igmp_version')

    # Permitir tiempo para que el controlador ODL detecte la topología
    sleep_time = 10
    print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para
permitir al controlador ODL descubrir la topología de red\033[0m'
    sleep(sleep_time)
```

```
88.
89.     print("\033[1;36m Red SDN con una Topología Lineal para
    Multidifusión con ODL\033[0m")
90.
91.     CLI(net)
92.     net.stop()
93.
94.
95. if __name__ == '__main__':
96.     setLogLevel( 'info' )
97.     # Generar topología y ejecutar escenario SDN
98.     topo = SDN_Topo()
99.     hosts = topo.get_host_list()
100.        SDN_Config(topo, hosts, False)
```

Anexo 2. *Script que incorpora una topología de red árbol con ODL.*
Fuente: *Elaborado por el Autor.*

```
1.  #!/usr/bin/env python
2.  # coding=utf-8
3.  from mininet.net import Mininet
4.  from mininet.topo import *
5.  from mininet.net import *
6.  from mininet.node import OVSSwitch, UserSwitch
7.  from mininet.link import TCLink
8.  from mininet.log import setLogLevel
9.  from mininet.cli import CLI
10. from mininet.node import Node, RemoteController
11. from time import sleep, time
12. from datetime import datetime
13. from subprocess import call
14.
15.
16. class SDN_Topo( Topo ):
17.     def __init__( self ):
18.         # Inicializar topología
19.         Topo.__init__( self )
20.
21.         # Añadir hosts y switches
22.         h1 = self.addHost('h1', ip='10.0.0.1')
23.         h2 = self.addHost('h2', ip='10.0.0.2')
24.         h3 = self.addHost('h3', ip='10.0.0.3')
25.         h4 = self.addHost('h4', ip='10.0.0.4')
26.         h5 = self.addHost('h5', ip='10.0.0.5')
27.         h6 = self.addHost('h6', ip='10.0.0.6')
28.         h7 = self.addHost('h7', ip='10.0.0.7')
29.         h8 = self.addHost('h8', ip='10.0.0.8')
30.         h9 = self.addHost('h9', ip='10.0.0.9')
```

```
        serv1 = self.addHost('serv1', ip='10.1.1.1')
        serv2 = self.addHost('serv2', ip='10.1.1.2')
        serv3 = self.addHost('serv3', ip='10.1.1.3')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        s6 = self.addSwitch('s6')
        s7 = self.addSwitch('s7')

        # Añadir enlaces (links)
        self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
        self.addLink(s1, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s2, s4, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s2, s5, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s3, s6, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s3, s7, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s4, h1, bw = 1000, use_htb = True)
        self.addLink(s4, h2, bw = 1000, use_htb = True)
        self.addLink(s5, h3, bw = 1000, use_htb = True)
        self.addLink(s5, h4, bw = 1000, use_htb = True)
        self.addLink(s6, h5, bw = 1000, use_htb = True)
        self.addLink(s6, h6, bw = 1000, use_htb = True)
        self.addLink(s7, h7, bw = 1000, use_htb = True)
        self.addLink(s7, h8, bw = 1000, use_htb = True)
        self.addLink(s7, h9, bw = 1000, use_htb = True)
        self.addLink(serv1, s2, bw = 1000, use_htb = True)
        self.addLink(serv2, s1, bw = 1000, use_htb = True)
        self.addLink(serv3, s3, bw = 1000, use_htb = True)


    def get_host_list(self):
        return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
    'serv3']

    def get_switch_list(self):
        return ['s1', 's2', 's3', 's4', 's5', 's6', 's7']


def SDN_Config(topo, hosts = [], interactive = False):

    # Configuración de red para usar el controlador externo ODL.
```

```
74.     net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
    link=TCLink, build=False, autoSetMacs=True)
75.     net.addController('odl', RemoteController, ip = '127.0.0.1', port = 6633)
76.     net.start()
77.
78.
79.     # Configurar interface de control en switches de red
80.     for switch_name in topo.get_switch_list():
81.         net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
82.         net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
83.         net.get('odl').cmd('route add -host ' + net.get(switch_name).IP() + '
    dev lo')
84.
85.
86.     # Configuración de hosts para soporte multicast con igmpv3
87.     for host_name in topo.get_host_list():
88.         # Configurar rutas multicast para interfaces virtuales
89.         net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
    eth0')
90.         # Configurar igmpv3 en cada host
91.         net.get(host_name).cmd('echo 3 >
    /proc/sys/net/ipv4/conf/all/force_igmp_version')
92.
93.
94.     # Permitir tiempo para que el controlador ODL detecte la topología
95.     sleep_time = 10
96.     print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para
    permitir al controlador ODL descubrir la topología de red\033[0m'
97.     sleep(sleep_time)
98.
99.     print("\033[1;36m Red SDN con una Topología Árbol para
    Multidifusión con ODL\033[0m")
100.
101.         CLI(net)
102.         net.stop()
103.
104.
105.     if __name__ == '__main__':
106.         setLogLevel( 'info' )
107.         # Generar topología y ejecutar escenario SDN
108.         topo = SDN_Topo()
109.         hosts = topo.get_host_list()
110.         SDN_Config(topo, hosts, False)
```

Anexo 3. *Script que incorpora una topología de red malla con ODL.*
Fuente: *Elaborado por el Autor.*

```
#!/usr/bin/env python
# coding=utf-8
from mininet.net import Mininet
from mininet.topo import *
from mininet.net import *
from mininet.node import OVSSwitch, UserSwitch
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import Node, RemoteController
from time import sleep, time
from datetime import datetime
from subprocess import call


class SDN_Topo( Topo ):
    def __init__( self ):
        # Inicializar topología
        Topo.__init__( self )

        # Añadir hosts y switches
        h1 = self.addHost('h1', ip='10.0.0.1')
        h2 = self.addHost('h2', ip='10.0.0.2')
        h3 = self.addHost('h3', ip='10.0.0.3')
        h4 = self.addHost('h4', ip='10.0.0.4')
        h5 = self.addHost('h5', ip='10.0.0.5')
        h6 = self.addHost('h6', ip='10.0.0.6')
        h7 = self.addHost('h7', ip='10.0.0.7')
        h8 = self.addHost('h8', ip='10.0.0.8')
        h9 = self.addHost('h9', ip='10.0.0.9')
        serv1 = self.addHost('serv1', ip='10.1.1.1')
        serv2 = self.addHost('serv2', ip='10.1.1.2')
        serv3 = self.addHost('serv3', ip='10.1.1.3')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        s6 = self.addSwitch('s6')
        s7 = self.addSwitch('s7')

        # Añadir enlaces (links)
        self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
```

```
45.         self.addLink(s2, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
46.         self.addLink(s3, s4, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
47.         self.addLink(s4, s1, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
48.         self.addLink(s1, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
49.         self.addLink(s2, s5, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
50.         self.addLink(s4, s7, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
51.         self.addLink(s3, s6, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
52.         self.addLink(s5, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
53.         self.addLink(s7, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
54.         self.addLink(s6, s5, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
55.         self.addLink(s6, s7, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
56.         self.addLink(s5, h1, bw = 1000, use_htb = True)
57.         self.addLink(s5, h2, bw = 1000, use_htb = True)
58.         self.addLink(s5, h3, bw = 1000, use_htb = True)
59.         self.addLink(s6, h4, bw = 1000, use_htb = True)
60.         self.addLink(s6, h5, bw = 1000, use_htb = True)
61.         self.addLink(s6, h6, bw = 1000, use_htb = True)
62.         self.addLink(s7, h7, bw = 1000, use_htb = True)
63.         self.addLink(s7, h8, bw = 1000, use_htb = True)
64.         self.addLink(s7, h9, bw = 1000, use_htb = True)
65.         self.addLink(serv1, s2, bw = 1000, use_htb = True)
66.         self.addLink(serv2, s1, bw = 1000, use_htb = True)
67.         self.addLink(serv3, s4, bw = 1000, use_htb = True)
68.
69.
70.     def get_host_list(self):
71.         return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
    'serv3']
72.
73.     def get_switch_list(self):
74.         return ['s1', 's2', 's3', 's4', 's5', 's6', 's7']
75.
76.
77. def SDN_Config(topo, hosts = [], interactive = False):
78.
79.     # Configuración de red para usar el controlador externo ODL.
80.     net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
    link=TCLink, build=False, autoSetMacs=True)
81.     net.addController('odl', RemoteController, ip = '127.0.0.1', port = 6633)
```

```
82.     net.start()
83.
84.
85.     # Configurar interface de control en switches de red
86.     for switch_name in topo.get_switch_list():
87.         net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
88.         net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
89.         net.get('odl').cmd('route add -host ' + net.get(switch_name).IP() + '
    dev lo')
90.
91.     # Configuración de hosts para soporte multicast con igmpv3
92.     for host_name in topo.get_host_list():
93.         # Configurar rutas multicast para interfaces virtuales
94.         net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
    eth0')
95.         # Configurar igmpv3 en cada host
96.         net.get(host_name).cmd('echo 3 >
    /proc/sys/net/ipv4/conf/all/force_igmp_version')
97.
98.
99.     # Permitir tiempo para que el controlador ODL detecte la topología
100.         sleep_time = 10
101.         print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para
    permitir al controlador ODL descubrir la topología de red\033[0m'
102.         sleep(sleep_time)
103.
104.         print("\033[1;36m Red SDN con una Topología Malla para
    Multidifusión con ODL\033[0m")
105.
106.         CLI(net)
107.         net.stop()
108.
109.
110.     if __name__ == '__main__':
111.         setLogLevel( 'info' )
112.         # Generar topología y ejecutar escenario SDN
113.         topo = SDN_Topo()
114.         hosts = topo.get_host_list()
115.         SDN_Config(topo, hosts, False)
```

Anexo 4. *Script que incorpora una topología de red lineal con POX.*
Fuente: *Elaborado por el Autor.*

```
1.  #!/usr/bin/env python
2.  # coding=utf-8
3.  from mininet.net import Mininet
4.  from mininet.topo import *
5.  from mininet.net import *
```

```
from mininet.node import OVSSwitch, UserSwitch
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import Node, RemoteController
from time import sleep, time
from datetime import datetime
from subprocess import call


class SDN_Topo( Topo ):
    def __init__( self ):
        # Inicializar topología
        Topo.__init__( self )

        # Añadir hosts y switches
        h1 = self.addHost('h1', ip='10.0.0.1')
        h2 = self.addHost('h2', ip='10.0.0.2')
        h3 = self.addHost('h3', ip='10.0.0.3')
        h4 = self.addHost('h4', ip='10.0.0.4')
        h5 = self.addHost('h5', ip='10.0.0.5')
        h6 = self.addHost('h6', ip='10.0.0.6')
        h7 = self.addHost('h7', ip='10.0.0.7')
        h8 = self.addHost('h8', ip='10.0.0.8')
        h9 = self.addHost('h9', ip='10.0.0.9')
        serv1 = self.addHost('serv1', ip='10.1.1.1')
        serv2 = self.addHost('serv2', ip='10.1.1.2')
        serv3 = self.addHost('serv3', ip='10.1.1.3')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')


        # Añadir enlaces (links)
        self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
        self.addLink(s2, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s1, h1, bw = 1000, use_htb = True)
        self.addLink(s1, h2, bw = 1000, use_htb = True)
        self.addLink(s1, h3, bw = 1000, use_htb = True)
        self.addLink(s2, h4, bw = 1000, use_htb = True)
        self.addLink(s2, h5, bw = 1000, use_htb = True)
        self.addLink(s2, h6, bw = 1000, use_htb = True)
        self.addLink(s3, h7, bw = 1000, use_htb = True)
        self.addLink(s3, h8, bw = 1000, use_htb = True)
        self.addLink(s3, h9, bw = 1000, use_htb = True)
        self.addLink(serv1, s1, bw = 1000, use_htb = True)
        self.addLink(serv2, s2, bw = 1000, use_htb = True)
```

```
54.         self.addLink(serv3, s3, bw = 1000, use_htb = True)
55.
56.
57.     def get_host_list(self):
58.         return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
    'serv3']
59.
60.     def get_switch_list(self):
61.         return ['s1', 's2', 's3']
62.
63.
64. def SDN_Config(topo, hosts = [], interactive = False):
65.
66.     # Configuración de red para usar el controlador externo POX.
67.     net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
    link=TCLink, build=False, autoSetMacs=True)
68.     net.addController('pox', RemoteController, ip = '127.0.0.1', port = 6633)
69.     net.start()
70.
71.
72.     # Configurar interface de control en switches de red
73.     for switch_name in topo.get_switch_list():
74.         net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
75.         net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
76.         net.get('pox').cmd('route add -host ' + net.get(switch_name).IP() + '
    dev lo')
77.
78.     # Configurar rutas multicast para interfaces virtuales
79.     for host_name in topo.get_host_list():
80.         net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
    eth0')
81.
82.
83.     # Permitir tiempo para que el controlador POX detecte la topología
84.     sleep_time = 10
85.     print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para
    permitir al controlador POX descubrir la topología de red\033[0m'
86.     sleep(sleep_time)
87.
88.     print("\033[1;36m Red SDN con una Topología Lineal para
    Multidifusión con POX\033[0m")
89.
90.     CLI(net)
91.     net.stop()
92.
93.
94. if __name__ == '__main__':
95.     setLogLevel( 'info' )
96.     # Generar topología y ejecutar escenario SDN
97.     topo = SDN_Topo()
```

```
    hosts = topo.get_host_list()
    SDN_Config(topo, hosts, False)
```

Anexo 5. *Script que incorpora una topología de red árbol con POX.*
Fuente: *Elaborado por el Autor.*

```
#!/usr/bin/env python
# coding=utf-8
from mininet.net import Mininet
from mininet.topo import *
from mininet.net import *
from mininet.node import OVSSwitch, UserSwitch
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import Node, RemoteController
from time import sleep, time
from datetime import datetime
from subprocess import call


class SDN_Topo( Topo ):
    def __init__( self ):
        # Inicializar topología
        Topo.__init__( self )

        # Añadir hosts y switches
        h1 = self.addHost('h1', ip='10.0.0.1')
        h2 = self.addHost('h2', ip='10.0.0.2')
        h3 = self.addHost('h3', ip='10.0.0.3')
        h4 = self.addHost('h4', ip='10.0.0.4')
        h5 = self.addHost('h5', ip='10.0.0.5')
        h6 = self.addHost('h6', ip='10.0.0.6')
        h7 = self.addHost('h7', ip='10.0.0.7')
        h8 = self.addHost('h8', ip='10.0.0.8')
        h9 = self.addHost('h9', ip='10.0.0.9')
        serv1 = self.addHost('serv1', ip='10.1.1.1')
        serv2 = self.addHost('serv2', ip='10.1.1.2')
        serv3 = self.addHost('serv3', ip='10.1.1.3')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        s6 = self.addSwitch('s6')
        s7 = self.addSwitch('s7')

```

```
        # Añadir enlaces (links)
        self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
        self.addLink(s1, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s2, s4, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s2, s5, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s3, s6, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s3, s7, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
        self.addLink(s4, h1, bw = 1000, use_htb = True)
        self.addLink(s4, h2, bw = 1000, use_htb = True)
        self.addLink(s5, h3, bw = 1000, use_htb = True)
        self.addLink(s5, h4, bw = 1000, use_htb = True)
        self.addLink(s6, h5, bw = 1000, use_htb = True)
        self.addLink(s6, h6, bw = 1000, use_htb = True)
        self.addLink(s7, h7, bw = 1000, use_htb = True)
        self.addLink(s7, h8, bw = 1000, use_htb = True)
        self.addLink(s7, h9, bw = 1000, use_htb = True)
        self.addLink(serv1, s2, bw = 1000, use_htb = True)
        self.addLink(serv2, s1, bw = 1000, use_htb = True)
        self.addLink(serv3, s3, bw = 1000, use_htb = True)


    def get_host_list(self):
        return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
    'serv3']

    def get_switch_list(self):
        return ['s1', 's2', 's3', 's4', 's5', 's6', 's7']


def SDN_Config(topo, hosts = [], interactive = False):

    # Configuración de red para usar el controlador externo POX.
    net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
    link=TCLink, build=False, autoSetMacs=True)
    net.addController('pox', RemoteController, ip = '127.0.0.1', port = 6633)
    net.start()


    # Configurar interface de control en switches de red
    for switch_name in topo.get_switch_list():
        net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
        net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
        net.get('pox').cmd('route add -host ' + net.get(switch_name).IP() + '
    dev lo')
```

```
84.
85.     # Configurar rutas multicast para interfaces virtuales
86.     for host_name in topo.get_host_list():
87.         net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
    eth0')
88.
89.
90.     # Permitir tiempo para que el controlador POX detecte la topología
91.     sleep_time = 10
92.     print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para
    permitir al controlador POX descubrir la topología de red\033[0m'
93.     sleep(sleep_time)
94.
95.     print("\033[1;36m Red SDN con una Topología Árbol para
    Multidifusión con POX\033[0m")
96.
97.     CLI(net)
98.     net.stop()
99.
100.
101.    if __name__ == '__main__':
102.        setLogLevel( 'info' )
103.        # Generar topología y ejecutar escenario SDN
104.        topo = SDN_Topo()
105.        hosts = topo.get_host_list()
106.        SDN_Config(topo, hosts, False)
```

Anexo 6. *Script que incorpora una topología de red malla con POX.*
Fuente: *Elaborado por el Autor.*

```
1.  #!/usr/bin/env python
2.  # coding=utf-8
3.  from mininet.net import Mininet
4.  from mininet.topo import *
5.  from mininet.net import *
6.  from mininet.node import OVSSwitch, UserSwitch
7.  from mininet.link import TCLink
8.  from mininet.log import setLogLevel
9.  from mininet.cli import CLI
10. from mininet.node import Node, RemoteController
11. from time import sleep, time
12. from datetime import datetime
13. from subprocess import call
14.
15.
16. class SDN_Topo( Topo ):
17.     def __init__( self ):
18.         # Inicializar topología
```

```
19.         Topo.__init__( self )
20.
21.         # Añadir hosts y switches
22.         h1 = self.addHost('h1', ip='10.0.0.1')
23.         h2 = self.addHost('h2', ip='10.0.0.2')
24.         h3 = self.addHost('h3', ip='10.0.0.3')
25.         h4 = self.addHost('h4', ip='10.0.0.4')
26.         h5 = self.addHost('h5', ip='10.0.0.5')
27.         h6 = self.addHost('h6', ip='10.0.0.6')
28.         h7 = self.addHost('h7', ip='10.0.0.7')
29.         h8 = self.addHost('h8', ip='10.0.0.8')
30.         h9 = self.addHost('h9', ip='10.0.0.9')
31.         serv1 = self.addHost('serv1', ip='10.1.1.1')
32.         serv2 = self.addHost('serv2', ip='10.1.1.2')
33.         serv3 = self.addHost('serv3', ip='10.1.1.3')
34.
35.         s1 = self.addSwitch('s1')
36.         s2 = self.addSwitch('s2')
37.         s3 = self.addSwitch('s3')
38.         s4 = self.addSwitch('s4')
39.         s5 = self.addSwitch('s5')
40.         s6 = self.addSwitch('s6')
41.         s7 = self.addSwitch('s7')
42.
43.         # Añadir enlaces (links)
44.         self.addLink(s1, s2, bw = 100, delay = '0ms', loss = 0, use_htb =
    True) # bw = Mbps, Packet loss = %
45.         self.addLink(s2, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
46.         self.addLink(s3, s4, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
47.         self.addLink(s4, s1, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
48.         self.addLink(s1, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
49.         self.addLink(s2, s5, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
50.         self.addLink(s4, s7, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
51.         self.addLink(s3, s6, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
52.         self.addLink(s5, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
53.         self.addLink(s7, s3, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
54.         self.addLink(s6, s5, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
55.         self.addLink(s6, s7, bw = 100, delay = '0ms', loss = 0, use_htb =
    True)
56.         self.addLink(s5, h1, bw = 1000, use_htb = True)
```

```
57.         self.addLink(s5, h2, bw = 1000, use_htb = True)
58.         self.addLink(s5, h3, bw = 1000, use_htb = True)
59.         self.addLink(s6, h4, bw = 1000, use_htb = True)
60.         self.addLink(s6, h5, bw = 1000, use_htb = True)
61.         self.addLink(s6, h6, bw = 1000, use_htb = True)
62.         self.addLink(s7, h7, bw = 1000, use_htb = True)
63.         self.addLink(s7, h8, bw = 1000, use_htb = True)
64.         self.addLink(s7, h9, bw = 1000, use_htb = True)
65.         self.addLink(serv1, s2, bw = 1000, use_htb = True)
66.         self.addLink(serv2, s1, bw = 1000, use_htb = True)
67.         self.addLink(serv3, s4, bw = 1000, use_htb = True)
68.
69.
70.     def get_host_list(self):
71.         return ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'serv1', 'serv2',
    'serv3']
72.
73.     def get_switch_list(self):
74.         return ['s1', 's2', 's3', 's4', 's5', 's6', 's7']
75.
76.
77. def SDN_Config(topo, hosts = [], interactive = False):
78.
79.     # Configuración de red para usar el controlador externo POX.
80.     net = Mininet(topo, controller=RemoteController, switch=OVSSwitch,
    link=TCLink, build=False, autoSetMacs=True)
81.     net.addController('pox', RemoteController, ip = '127.0.0.1', port = 6633)
82.     net.start()
83.
84.
85.     # Configurar interface de control en switches de red
86.     for switch_name in topo.get_switch_list():
87.         net.get(switch_name).controlIntf = net.get(switch_name).intf('lo')
88.         net.get(switch_name).cmd('route add -host 127.0.0.1 dev lo')
89.         net.get('pox').cmd('route add -host ' + net.get(switch_name).IP() + '
    dev lo')
90.
91.     # Configurar rutas multicast para interfaces virtuales
92.     for host_name in topo.get_host_list():
93.         net.get(host_name).cmd('route add -net 224.0.0.0/4 ' + host_name + '-
    eth0')
94.
95.
96.     # Permitir tiempo para que el controlador POX detecte la topología
97.     sleep_time = 10
98.     print '\033[1;32m Esperando ' + str(sleep_time) + ' segundos para
    permitir al controlador POX descubrir la topología de red\033[0m'
99.     sleep(sleep_time)
100.
```

```
101.        print("\033[1;36m Red SDN con una Topología Malla para
    Multidifusión con POX\033[0m")
102.
103.        CLI(net)
104.        net.stop()
105.
106.
107.    if __name__ == '__main__':
108.        setLogLevel( 'info' )
109.        # Generar topología y ejecutar escenario SDN
110.        topo = SDN_Topo()
111.        hosts = topo.get_host_list()
112.        SDN_Config(topo, hosts, False)
```

Anexo 7. *Script que integra parámetros de retardo y pérdida de paquetes en la topología de red lineal.*

Fuente: *Elaborado por el Autor.*

```
1.          # Añadir enlaces (links)
2.          self.addLink(s1, s2, bw = 100, delay = '10ms', loss = 5, use_htb =
    True) # bw = Mbps, Packet loss = %
3.          self.addLink(s2, s3, bw = 100, delay = '5ms', loss = 10, use_htb =
    True)
4.          self.addLink(s1, h1, bw = 1000, use_htb = True)
5.          self.addLink(s1, h2, bw = 1000, use_htb = True)
6.          self.addLink(s1, h3, bw = 1000, use_htb = True)
7.          self.addLink(s2, h4, bw = 1000, use_htb = True)
8.          self.addLink(s2, h5, bw = 1000, use_htb = True)
9.          self.addLink(s2, h6, bw = 1000, use_htb = True)
10.         self.addLink(s3, h7, bw = 1000, use_htb = True)
11.         self.addLink(s3, h8, bw = 1000, use_htb = True)
12.         self.addLink(s3, h9, bw = 1000, use_htb = True)
13.         self.addLink(serv1, s1, bw = 1000, use_htb = True)
14.         self.addLink(serv2, s2, bw = 1000, use_htb = True)
15.         self.addLink(serv3, s3, bw = 1000, use_htb = True)
```

Anexo 8. *Script que integra parámetros de retardo y pérdida de paquetes en la topología de red árbol.*

Fuente: *Elaborado por el Autor.*

```
1.          # Añadir enlaces (links)
2.          self.addLink(s1, s2, bw = 100, delay = '1ms', loss = 1, use_htb =
    True) # bw = Mbps, Packet loss = %
3.          self.addLink(s1, s3, bw = 100, delay = '1ms', loss = 1, use_htb =
    True)
```

```
        self.addLink(s2, s4, bw = 100, delay = '5ms', loss = 10, use_htb = True)
        self.addLink(s2, s5, bw = 100, delay = '10ms', loss = 5, use_htb = True)
        self.addLink(s3, s6, bw = 100, delay = '5ms', loss = 10, use_htb = True)
        self.addLink(s3, s7, bw = 100, delay = '10ms', loss = 5, use_htb = True)
        self.addLink(s4, h1, bw = 1000, use_htb = True)
        self.addLink(s4, h2, bw = 1000, use_htb = True)
        self.addLink(s5, h3, bw = 1000, use_htb = True)
        self.addLink(s5, h4, bw = 1000, use_htb = True)
        self.addLink(s6, h5, bw = 1000, use_htb = True)
        self.addLink(s6, h6, bw = 1000, use_htb = True)
        self.addLink(s7, h7, bw = 1000, use_htb = True)
        self.addLink(s7, h8, bw = 1000, use_htb = True)
        self.addLink(s7, h9, bw = 1000, use_htb = True)
        self.addLink(serv1, s2, bw = 1000, use_htb = True)
        self.addLink(serv2, s1, bw = 1000, use_htb = True)
        self.addLink(serv3, s3, bw = 1000, use_htb = True)
```

Anexo 9. *Script que integra parámetros de retardo y pérdida de paquetes en la topología de red malla.*

Fuente: *Elaborado por el Autor.*

```
        # Añadir enlaces (links)
        self.addLink(s1, s2, bw = 100, delay = '1ms', loss = 10, use_htb = True) # bw = Mbps, Packet loss = %
        self.addLink(s2, s3, bw = 100, delay = '3ms', loss = 5, use_htb = True)
        self.addLink(s3, s4, bw = 100, delay = '5ms', loss = 3, use_htb = True)
        self.addLink(s4, s1, bw = 100, delay = '10ms', loss = 1, use_htb = True)
        self.addLink(s1, s3, bw = 100, delay = '1ms', loss = 10, use_htb = True)
        self.addLink(s2, s5, bw = 100, delay = '3ms', loss = 5, use_htb = True)
        self.addLink(s4, s7, bw = 100, delay = '5ms', loss = 3, use_htb = True)
        self.addLink(s3, s6, bw = 100, delay = '10ms', loss = 1, use_htb = True)
        self.addLink(s5, s3, bw = 100, delay = '1ms', loss = 10, use_htb = True)
        self.addLink(s7, s3, bw = 100, delay = '3ms', loss = 5, use_htb = True)
        self.addLink(s6, s5, bw = 100, delay = '5ms', loss = 3, use_htb = True)
```

```
13.       self.addLink(s6, s7, bw = 100, delay = '10ms', loss = 1, use_htb =
    True)
14.       self.addLink(s5, h1, bw = 1000, use_htb = True)
15.       self.addLink(s5, h2, bw = 1000, use_htb = True)
16.       self.addLink(s5, h3, bw = 1000, use_htb = True)
17.       self.addLink(s6, h4, bw = 1000, use_htb = True)
18.       self.addLink(s6, h5, bw = 1000, use_htb = True)
19.       self.addLink(s6, h6, bw = 1000, use_htb = True)
20.       self.addLink(s7, h7, bw = 1000, use_htb = True)
21.       self.addLink(s7, h8, bw = 1000, use_htb = True)
22.       self.addLink(s7, h9, bw = 1000, use_htb = True)
23.       self.addLink(serv1, s2, bw = 1000, use_htb = True)
24.       self.addLink(serv2, s1, bw = 1000, use_htb = True)
25.       self.addLink(serv3, s4, bw = 1000, use_htb = True)
```

Anexo 10. *Especificaciones de Switch SDN de hardware SEL-2742S*
Fuente: *Extraído del sitio web SEL Schweitzer Engineering Laboratories.*

SEL-2742S Specifications

General	
Communications Ports	Ethernet ports: 13 **Data Rates** One 10/100 Mbps ETH F management port Four 100/1000 Mbps ports Eight 10/100 Mbps ports **Connectors** RJ45 female or LC fiber (multimode or single-mode) **Standard** IEEE 802.3, OpenFlow 1.3
Power Supply[1]	**DC Power Supply** Rated voltage range: 24/48 Vdc Min/max voltage range: 9.6–60.0 Vdc Maximum burden: <45 W **24/48 Volt Power Supply** 4.0 A, 150 Vdc time-lag T, 250 Vac/1,500 A break rating PoE: 30 W per port, max two ports
Dimensions and Weight[2]	Height: 200.66 mm (7.9 in) Depth: 139.7 mm (5.5 in) Width: 69.85 mm (2.75 in) Weight: 1.81 kg (4 lb)
Environmental and Compliance	Operating temperature: –40° to +85°C (–40° to +185°F) Designed and manufactured under an ISO 9001-certified quality management system UL-recognized to U.S. and Canadian safety standards 47 CFR 15B, FCC Class A CE mark