Marco SDN seguro y de bajo costo para trazabilidad IoT en cadenas agroalimentarias

Daniel Botero Parra

Asesor

Raúl Bareño Gutiérrez

Universidad Nacional Abierta y a Distancia UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería ECBTI

Especialización en Redes de Telecomunicaciones

2026

Dedicatoria

Dedico este trabajo a mi familia, por su apoyo incondicional y por ser mi motivación constante para seguir creciendo profesional y personalmente. A quienes han confiado en mí y han impulsado mi interés por la tecnología, la innovación y la mejora continua.

Agradecimientos

Agradezco a la Universidad Nacional Abierta y a Distancia – UNAD y a la Escuela de Ciencias Básicas, Tecnología e Ingeniería por la formación brindada.

A los docentes y asesores que orientaron este proceso académico, y a todas las personas que, desde el ámbito profesional y técnico, aportaron conocimientos y experiencias que enriquecieron este trabajo.

Resumen

La trazabilidad en las cadenas agroalimentarias enfrenta limitaciones relacionadas con la gestión del tráfico de datos, la escalabilidad de las redes y la seguridad de la información, especialmente en entornos donde se integran múltiples dispositivos IoT. Estas dificultades afectan la confiabilidad de los sistemas y la toma de decisiones en procesos productivos y logísticos. En este contexto, resulta necesario desarrollar soluciones tecnológicas que permitan mejorar la administración de la red, garantizar la integridad de los datos y facilitar la adopción de estas tecnologías en entornos reales.

El presente trabajo tiene como objetivo diseñar un marco basado en Software Defined Networking (SDN) seguro y de bajo costo para la trazabilidad IoT en cadenas agroalimentarias. Para ello, se emplea una metodología mixta que combina el análisis cualitativo de literatura científica con la validación cuantitativa mediante simulación de red.

Como resultado, se establecen lineamientos para una arquitectura SDN-IoT que permite mejorar la gestión del tráfico, fortalecer los mecanismos de seguridad y optimizar el uso de recursos en sistemas de trazabilidad. Finalmente, se concluye que la integración de SDN e IoT constituye una alternativa viable para el sector agroalimentario, al permitir el desarrollo de soluciones más eficientes, seguras y accesibles, contribuyendo a la transformación digital de estos sistemas productivos.

Palabras clave: SDN, IoT, trazabilidad agroalimentaria, seguridad de redes, cadena de suministro

Abstract

Traceability in agri-food supply chains faces limitations related to data traffic management, network scalability, and information security, especially in environments where multiple IoT devices are integrated. These challenges affect the reliability of systems and decision-making in production and logistics processes. In this context, it is necessary to develop technological solutions that improve network management, ensure data integrity, and facilitate the adoption of these technologies in real-world environments.

The objective of this work is to design a secure and low-cost Software Defined Networking (SDN)-based framework for IoT traceability in agri-food supply chains. To achieve this, a mixed methodology is employed, combining qualitative analysis of scientific literature with quantitative validation through network simulation.

As a result, guidelines are established for an SDN-IoT architecture that improves traffic management, strengthens security mechanisms, and optimizes resource usage in traceability systems. Finally, it is concluded that the integration of SDN and IoT constitutes a viable alternative for the agri-food sector, enabling the development of more efficient, secure, and accessible solutions, and contributing to the digital transformation of these production systems.

Keywords: SDN, IoT, agri-food traceability, network security, supply chain

Tabla de Contenido

Objetivos 16

Marco Referencial 17

Diseño Metodológico 21

Resultados 23

Objetivo Específico 2 31

Objetivo Específico 3 38

Escenario 1 Red sin SDN 71

Escenario 2 Red con SDN 82

Escenario 3 Validación de Seguridad 96

Análisis de Resultados 101

Conclusiones 104

Referencias Bibliográficas 106

Lista de Tablas

Tabla 1 *Matriz comparativa de arquitecturas y mecanismos de seguridad en redes SDN-IoT* .. 28

Tabla 2 *Subnetting de la red* 41

Tabla 3 *Asignación de prioridades* 89

Tabla 4 *Comparación de resultados de escenarios 1 y 2* 101

Lista de Figuras

Figura 1 *Estructura por capas de marco arquitectónico* 32

Figura 2 *Arquitectura SDN/IoT* 35

Figura 3 *Simulación en GNS3 de la arquitectura* 40

Figura 4 *Interfaces SW1* 42

Figura 5 *Creación de Bridge con las interfaces* 43

Figura 6 *Validación de configuración SW1* 44

Figura 7 *Interfaces SW2* 45

Figura 8 *Validación de configuración SW2* 46

Figura 9 *Interfaces SW3* 47

Figura 10 *Validación de configuración SW3* 48

Figura 11 *Configuración interfaz Servidor IoT* 49

Figura 12 *Configuración interfaz Controlador SDN* 50

Figura 13 *Configuración interfaz Servidor de trazabilidad* 51

Figura 14 *Ping de Servidor IoT a SW1* 52

Figura 15 *Ping de SW2 a SW3* 53

Figura 16 *Ping de Servidor IoT a Controlador SDN* 54

Figura 17 *Ping de Servidor IoT a Servidor de trazabilidad* 55

Figura 18 *Configuración OpenFlow* 56

Figura 19 *Entorno OpenFlow en controlador SDN* 57

Figura 20 *Conexiones en OpenFlow* 58

Figura 21 E*ventos EventOFPPacketIn* 60

Figura 22 *Flows instalados SW1* 61

Figura 23 *Tabla PostgreSQL* 62

Figura 24 *API de Servidor de trazabilidad* 63

Figura 25 *Ejecución de código para recibir datos* 64

Figura 26 *Código Python para envío de datos* 66

Figura 27 *Envío de datos desde el servidor IoT* 67

Figura 28 *Código de Dashboard* 68

Figura 29 *Dashboard funcionando por puerto 5000* 70

Figura 30 *Prueba inicial de Ping y envío de datos desde Servidor IoT* 71

Figura 31 *Código modificado para envío de datos por HTTP con porcentaje de error* 73

Figura 32 *Prueba inicial sin carga de red* 74

Figura 33 *Habilitación de servidores* 76

Figura 34 *Resultados sin SDN en servidor IoT* 79

Figura 35 *Resultados sin SDN en servidor de trazabilidad* 80

Figura 36 *Resultados sin SDN en controlador SDN* 81

Figura 37 *Inicialización del controlador SDN* 83

Figura 38 *Parte 1. Código qos_controller.py* 85

Figura 39 *Parte 2. Código qos_controller.py* 86

Figura 40 *Parte 3. Código qos_controller.py* 88

Figura 41 *Eliminación de reglas y prueba de ping* 90

Figura 42 *Respuesta de Controlador SDN con aplicación de políticas* 91

Figura 43 *Resultados con SDN en servidor IoT* 93

Figura 44 *Resultados con SDN en servidor de trazabilidad* 94

Figura 45 *Resultados con SDN en controlador SDN* 95

Figura 46 *Red con PC conectado a SW1* 96

Figura 47 *Dirección IP asignada a PC* 97

Figura 48 *Codigo de qos_controller.py modificado* 98

Figura 49 *Prueba de mitigación de ataque*... 99

Introducción

La transformación digital ha impulsado la adopción de tecnologías emergentes en diversos sectores productivos, entre ellos el agroalimentario, donde la trazabilidad se ha convertido en un elemento clave para garantizar la calidad, seguridad y transparencia de los productos a lo largo de la cadena de suministro. En este contexto, el Internet de las Cosas (IoT) permite la recolección continua de datos en tiempo real mediante sensores distribuidos, facilitando el monitoreo de variables críticas durante los procesos de producción, almacenamiento y transporte (Rahman et al., 2022; Rossi et al., 2025).

No obstante, la integración de dispositivos IoT en infraestructuras de red plantea desafíos importantes relacionados con la gestión del tráfico, la escalabilidad y la seguridad de la información. La creciente cantidad de dispositivos conectados incrementa la complejidad de la red y amplía la superficie de ataque, lo que hace necesario implementar mecanismos que permitan garantizar la disponibilidad, integridad y confidencialidad de los datos. En este sentido, las redes definidas por software (SDN) han emergido como una alternativa que permite mejorar la administración de la red mediante la centralización del control y la implementación de políticas dinámicas de gestión del tráfico (Kreutz et al., 2015; Singh et al., 2019).

En este contexto, diversos estudios han analizado la evolución de las arquitecturas SDN aplicadas a entornos IoT, destacando su capacidad para adaptarse a escenarios altamente dinámicos y distribuidos. En particular, se ha evidenciado que la separación del plano de control y el plano de datos facilita la implementación de mecanismos avanzados de gestión de red, permitiendo optimizar el rendimiento y mejorar la eficiencia en el uso de los recursos disponibles (Zhang et al., 2023). Asimismo, se ha señalado que la integración de SDN en sistemas IoT

contribuye a mejorar la interoperabilidad entre dispositivos heterogéneos, lo cual resulta fundamental en entornos agroalimentarios donde coexisten múltiples tecnologías y protocolos.

De manera complementaria, el uso de tecnologías como la virtualización de funciones de red (NFV) y la orquestación de servicios ha permitido ampliar las capacidades de las arquitecturas SDN-IoT, facilitando la implementación de funciones de seguridad y control de tráfico de manera flexible y escalable. Estas estrategias permiten adaptar dinámicamente la red a las condiciones del entorno, mejorando la capacidad de respuesta frente a variaciones en la carga de tráfico o posibles fallos en la infraestructura (Zhou et al., 2024). Este tipo de enfoques resulta especialmente relevante en aplicaciones de trazabilidad, donde la continuidad y confiabilidad de los datos son factores críticos.

Adicionalmente, diversas investigaciones han abordado la necesidad de diseñar arquitecturas que no solo sean seguras y eficientes, sino también viables desde el punto de vista económico. En este contexto, se han propuesto enfoques basados en el uso de tecnologías de bajo costo y soluciones de código abierto que facilitan la adopción de SDN en entornos IoT, especialmente en aplicaciones donde los recursos son limitados. Asimismo, se han desarrollado mecanismos de seguridad orientados a la detección de amenazas y la mitigación de ataques, contribuyendo a mejorar la resiliencia de las redes en entornos distribuidos (Saini & Gupta, 2023; Rahman et al., 2021).

A partir de estas consideraciones, el presente trabajo se enfoca en el diseño de un marco basado en SDN seguro y de bajo costo para sistemas de trazabilidad IoT en cadenas agroalimentarias, con el propósito de mejorar la gestión del tráfico de red, fortalecer la seguridad de la información y facilitar la implementación de estas tecnologías en contextos productivos reales.

Justificación

La creciente digitalización del sector agroalimentario ha generado la necesidad de implementar sistemas de trazabilidad más eficientes, capaces de garantizar la calidad, seguridad y transparencia de los productos a lo largo de la cadena de suministro. En este contexto, la incorporación de tecnologías como el Internet de las Cosas permite la captura continua de datos en tiempo real; sin embargo, también introduce desafíos relacionados con la gestión del tráfico de red, la escalabilidad de los sistemas y la protección de la información, factores que impactan directamente la confiabilidad de los procesos productivos y logísticos.

Ante este escenario, la integración de redes definidas por software con entornos IoT surge como una alternativa que permite mejorar el control, la visibilidad y la administración de la infraestructura de red. La capacidad de centralizar la gestión, aplicar políticas dinámicas y optimizar el flujo de datos resulta especialmente adecuada para sistemas distribuidos, como los presentes en las cadenas agroalimentarias. En este sentido, el trabajo se enfoca en el diseño de un marco conceptual orientado a la trazabilidad IoT, considerando aspectos clave como la seguridad de la información, la gestión del tráfico y la eficiencia operativa.

La propuesta se limita al planteamiento de lineamientos estructurados basados en tecnologías accesibles y soluciones de bajo costo, priorizando su aplicabilidad en pequeñas y medianas agroindustrias. Asimismo, el uso de plataformas abiertas y el enfoque metodológico basado en análisis documental permiten desarrollar una solución viable sin requerir inversiones elevadas en infraestructura, favoreciendo su implementación en entornos productivos reales.

Planteamiento del Problema

Las cadenas agroalimentarias modernas enfrentan dificultades significativas para garantizar la trazabilidad, seguridad y transparencia de los productos desde su origen hasta el consumidor final. La falta de visibilidad en los procesos logísticos y productivos genera riesgos asociados a contaminación, fraude alimentario y pérdidas económicas. Estudios sobre trazabilidad alimentaria muestran que la ausencia de sistemas integrados dificulta la toma de decisiones y reduce la confianza del consumidor (Zhang et al., 2020). Asimismo, la digitalización de estos procesos requiere soluciones tecnológicas que permitan registrar y verificar datos en tiempo real sin incrementar significativamente los costos operativos.

El uso de tecnologías IoT ha permitido mejorar la recolección de datos en tiempo real dentro de las cadenas de suministro; sin embargo, estos entornos presentan vulnerabilidades relacionadas con la seguridad, privacidad y gestión eficiente del tráfico de red (Ammar et al., 2018). En este contexto, las arquitecturas basadas en Software Defined Networking (SDN) emergen como una alternativa para optimizar la administración de redes IoT al proporcionar control centralizado, flexibilidad y capacidad de automatización (Kreutz et al., 2015). No obstante, investigaciones recientes destacan que las redes SDN-IoT aún enfrentan desafíos relacionados con ataques DDoS, autenticación segura y protección de datos sensibles (Luo et al., 2019).

Adicionalmente, la convergencia entre IoT, SDN y tecnologías emergentes como blockchain y computación en el borde plantea oportunidades para mejorar la seguridad y confiabilidad de los sistemas de trazabilidad. Se ha demostrado que la integración de blockchain puede fortalecer la integridad de los datos y la confianza entre los actores de la cadena de suministro (Tian, 2017), mientras que los enfoques basados en edge computing y orquestación

inteligente de servicios optimizan la latencia y eficiencia operativa en entornos IoT (Qu et al., 2020). Sin embargo, estas soluciones suelen implicar altos costos de implementación y complejidad técnica, lo que limita su adopción en contextos agroalimentarios de pequeña y mediana escala.

En consecuencia, existe la necesidad de diseñar un marco tecnológico seguro, escalable y de bajo costo que integre SDN e IoT para mejorar la trazabilidad en cadenas agroalimentarias, garantizando la protección de datos, la confiabilidad del sistema y la viabilidad económica de su implementación. Este problema justifica el desarrollo de un modelo que combine control de red inteligente, mecanismos de seguridad y monitoreo en tiempo real, adaptado a las necesidades del sector agrícola.

En este contexto, surge la siguiente pregunta de investigación.

¿Cómo puede estructurarse un marco basado en Software Defined Networking (SDN) seguro y de bajo costo que permita fortalecer la trazabilidad mediante IoT en cadenas agroalimentarias, a partir del análisis de arquitecturas y mecanismos de seguridad reportados en la literatura científica?

Objetivos

Objetivo General

Diseñar un marco SDN de bajo costo para la trazabilidad IoT en cadenas agroalimentarias, optimizando la gestión del tráfico y fortaleciendo la seguridad de la información.

Objetivos Específicos

Analizar arquitecturas y mecanismos de seguridad en redes SDN-IoT reportados en la literatura científica, con el fin de identificar ventajas, limitaciones y condiciones de implementación aplicables a sistemas de trazabilidad agroalimentaria.

Diseñar un marco arquitectónico basado en SDN que integre mecanismos de gestión de tráfico, control de acceso y monitoreo de seguridad para redes IoT orientadas a la trazabilidad en cadenas agroalimentarias.

Validar experimentalmente los principios del marco propuesto mediante la simulación de una red IoT basada en SDN, evaluando métricas de gestión del tráfico, latencia, congestión y capacidad de mitigación de ataques.

Marco Referencial

Antecedentes

La digitalización de las cadenas agroalimentarias ha impulsado el uso del Internet de las Cosas (IoT) para monitorear condiciones ambientales, localización y estado de los productos en tiempo real, mejorando la seguridad y calidad alimentaria (Rahman et al., 2022; Rossi et al., 2025). Estas soluciones permiten registrar información crítica durante el transporte y almacenamiento, facilitando la trazabilidad y la toma de decisiones logísticas.

Sin embargo, la integración masiva de dispositivos IoT genera desafíos relacionados con la escalabilidad, gestión del tráfico y seguridad de la red. El paradigma de Software Defined Networking (SDN) ha sido propuesto como una alternativa eficiente al permitir el control centralizado, la programación dinámica y la optimización del flujo de datos (Kreutz et al., 2015; Saini & Gupta, 2023).

Adicionalmente, estudios recientes destacan la necesidad de fortalecer la seguridad en entornos IoT, debido a amenazas como ataques DDoS, accesos no autorizados y manipulación de datos (Rahman et al., 2021; Singh et al., 2019). En este contexto, la integración SDN-IoT emerge como una solución para mejorar la resiliencia y confiabilidad de los sistemas digitales.

Marco Conceptual

Trazabilidad agroalimentaria: capacidad de rastrear productos desde su origen hasta el consumidor final para garantizar calidad y seguridad alimentaria (Rossi et al., 2025).

Internet de las Cosas (IoT): red de dispositivos interconectados que recopilan y transmiten datos mediante sensores inteligentes (Rahman et al., 2022).

Software Defined Networking (SDN): arquitectura de red que separa el plano de control del plano de datos para facilitar la gestión centralizada y programable (Kreutz et al., 2015).

Seguridad en redes IoT: conjunto de mecanismos que protegen la integridad, confidencialidad y disponibilidad de los datos (Singh et al., 2019).

Según la Food and Agriculture Organization of the United Nations (FAO, 2019), la cadena de suministro agroalimentaria comprende los procesos de producción, almacenamiento, distribución y comercialización de alimentos.

Marco Teórico

El enfoque de Software Defined Networking (SDN) permite la gestión centralizada del tráfico, la implementación de políticas de seguridad dinámicas y la optimización del rendimiento de la red, especialmente en entornos con alta densidad de dispositivos conectados. Esta arquitectura separa el plano de control del plano de datos, lo que facilita la administración programable de la infraestructura y mejora la eficiencia en la gestión de redes complejas (Kreutz et al., 2015; Zhang et al., 2023).

En entornos de Internet de las Cosas (IoT), la recopilación continua de datos mediante sensores permite la supervisión en tiempo real de variables críticas relacionadas con la producción, almacenamiento y transporte de productos. Estas capacidades tecnológicas contribuyen a optimizar los procesos logísticos y reducir pérdidas en cadenas de suministro, particularmente en el caso de productos perecederos (Rahman et al., 2022; Rossi et al., 2025).

Asimismo, diferentes investigaciones han explorado el uso de técnicas avanzadas como Service Function Chaining (SFC) y algoritmos de aprendizaje profundo para mejorar la eficiencia, resiliencia y gestión dinámica de los servicios de red en entornos SDN-IoT (Fu et al., 2020; Wang et al., 2021; Zhou et al., 2024). Estas estrategias permiten optimizar la asignación de recursos y mejorar la capacidad de respuesta frente a cambios en el tráfico de red o fallos en la infraestructura.

Por otra parte, algunos estudios han propuesto la integración de tecnologías emergentes como blockchain para reforzar la integridad y trazabilidad de los datos generados en entornos IoT distribuidos (Rahman et al., 2022). No obstante, en el presente trabajo estas tecnologías se consideran enfoques complementarios dentro del análisis de la literatura, mientras que el eje central del estudio se concentra en la arquitectura SDN y en los mecanismos de seguridad aplicables a sistemas IoT, con especial énfasis en soluciones viables y de bajo costo para su adopción en cadenas agroalimentarias.

Finalmente, los modelos de seguridad en redes definidas por software enfatizan la implementación de mecanismos como autenticación, cifrado, control de acceso y monitoreo del tráfico, los cuales permiten prevenir ataques y garantizar la integridad de la información en entornos IoT altamente distribuidos (Singh et al., 2019; Rao & Das, 2024). Estos enfoques constituyen la base conceptual para el análisis de arquitecturas seguras que permitan mejorar la confiabilidad de los sistemas de trazabilidad digital.

Marco Legal

La trazabilidad alimentaria está vinculada a normativas de seguridad sanitaria que buscan garantizar la inocuidad de los productos y la protección del consumidor. Organismos internacionales promueven la adopción de tecnologías digitales para mejorar la transparencia en la cadena alimentaria (FAO, 2019).

Asimismo, la implementación de soluciones IoT debe cumplir con regulaciones sobre protección de datos y seguridad de la información, asegurando el manejo adecuado de la información recolectada (Secure Access Control Protocol, 2020).

Las políticas de transformación digital fomentan la adopción de tecnologías emergentes para mejorar la competitividad del sector agroindustrial y fortalecer la seguridad de las infraestructuras críticas (Rahman et al., 2021).

Marco Contextual

El sector agroalimentario enfrenta desafíos relacionados con pérdidas post-cosecha, falta de visibilidad en la cadena logística y limitada digitalización de procesos, lo cual afecta la calidad del producto y la confianza del consumidor (FAO, 2019).

En regiones rurales y economías emergentes, las limitaciones de conectividad y recursos económicos dificultan la implementación de soluciones tecnológicas avanzadas, por lo que se requieren arquitecturas de bajo costo y alta eficiencia (Saini & Gupta, 2023; Digital Agriculture Transformation, 2019).

La integración de IoT con SDN ofrece una alternativa viable para mejorar la trazabilidad, optimizar la logística y fortalecer la seguridad de la información, contribuyendo a la modernización del sector agroalimentario (Fu et al., 2020; Rahman et al., 2022; Zhang et al., 2023).

Diseño Metodológico

La presente investigación se desarrolla bajo un enfoque de tipo aplicada con alcance descriptivo analítico, orientado al diseño de un marco basado en Software Defined Networking (SDN) para la trazabilidad IoT en cadenas agroalimentarias. Este enfoque permite analizar soluciones tecnológicas existentes y establecer lineamientos estructurados que respondan a problemáticas relacionadas con la gestión del tráfico, la seguridad de la información y la viabilidad de implementación en entornos reales. La investigación se fundamenta en la integración de conceptos asociados a redes SDN, Internet de las Cosas (IoT) y mecanismos de seguridad en redes distribuidas (Li et al., 2024; Alsmadi & Xu, 2019).

Para el desarrollo del estudio se emplea el análisis documental sistemático, el cual consiste en la identificación, selección, evaluación y síntesis de literatura científica relevante. Este método permite consolidar conocimiento en áreas tecnológicas emergentes y establecer un panorama claro sobre las soluciones existentes en arquitecturas SDN-IoT, especialmente en aspectos como control del tráfico, escalabilidad y seguridad (Awasthi & Sharma, 2024; Saini & Gupta, 2023). A partir de este análisis, se identifican tecnologías clave como protocolos de control de acceso, mecanismos de mitigación de ataques y estrategias de gestión de red que contribuyen al diseño de sistemas más robustos.

El proceso de análisis incluye la revisión comparativa de diferentes enfoques reportados en la literatura, considerando variables como arquitectura de red, mecanismos de seguridad, eficiencia en la gestión del tráfico, escalabilidad y viabilidad económica. En este sentido, se analizan propuestas orientadas a la detección y mitigación de ataques en entornos SDN-IoT, así como soluciones enfocadas en la autenticación de dispositivos y el control de acceso a los

recursos de red, lo cual permite identificar fortalezas y limitaciones en cada enfoque (Singh et al., 2019; Bhatia et al., 2024).

Con base en la información recopilada, se construye una matriz comparativa que permite organizar y evaluar las diferentes soluciones tecnológicas, facilitando la identificación de patrones, tendencias y oportunidades de mejora. Este análisis sirve como base para la definición de los lineamientos del marco propuesto, el cual integra capacidades de gestión del tráfico, mecanismos de seguridad y criterios de bajo costo, orientados a su aplicación en sistemas de trazabilidad agroalimentaria.

Finalmente, como parte de la validación del modelo propuesto, se plantea una fase de evaluación experimental mediante simulación de red, con el fin de analizar el comportamiento de la arquitectura SDN en un entorno controlado. Esta simulación permite evaluar aspectos como la gestión del tráfico, la respuesta ante condiciones de congestión y la efectividad de los mecanismos de control implementados, proporcionando evidencia sobre la viabilidad técnica de la propuesta. Asimismo, se consideran métricas de desempeño relacionadas con la eficiencia, la latencia y la estabilidad del sistema, en concordancia con enfoques actuales de optimización en redes SDN-IoT (Zhou et al., 2024).

Resultados

Objetivo Específico 1

Las redes definidas por software representan un paradigma emergente en el diseño y gestión de redes, caracterizado por la separación del plano de control y el plano de datos. En este modelo, el plano de control se centraliza en un controlador lógico que administra las decisiones de encaminamiento, mientras que el plano de datos se encarga exclusivamente del reenvío de paquetes. Esta arquitectura permite una gestión más flexible, programable y adaptable de la infraestructura de red, facilitando además la incorporación de mecanismos inteligentes para la gestión dinámica de recursos y servicios en entornos distribuidos (Li et al., 2024).

Uno de los principales beneficios de SDN es la capacidad de simplificar la administración de redes complejas mediante la centralización del control y la automatización de políticas de red (Kreutz et al., 2015). Esta separación facilita la implementación de mecanismos de seguridad, optimización del tráfico y monitoreo en tiempo real, lo cual resulta particularmente relevante en entornos donde se requiere gestionar grandes volúmenes de dispositivos conectados. En este sentido, diversos enfoques han propuesto modelos de orquestación confiable de servicios en redes SDN, orientados a fortalecer la seguridad interna y el control autónomo de la infraestructura mediante esquemas basados en confianza (Wang et al., 2022).

En paralelo, el Internet de las Cosas (IoT) ha emergido como una de las tecnologías clave en la transformación digital de múltiples sectores productivos. IoT se basa en la interconexión de dispositivos físicos capaces de recolectar, procesar y transmitir información a través de redes de comunicación. Estos dispositivos incluyen sensores, actuadores y sistemas embebidos que permiten monitorear condiciones ambientales, procesos industriales y operaciones logísticas en tiempo real (Awasthi & Sharma, 2024).

La integración entre SDN e IoT ha generado un nuevo enfoque arquitectónico conocido como redes SDN-IoT, en el cual la programabilidad de la red facilita la gestión dinámica de dispositivos distribuidos y flujos de datos generados por sensores inteligentes. En este contexto, SDN permite mejorar la escalabilidad, eficiencia y control de las infraestructuras IoT mediante la implementación de políticas de red centralizadas y mecanismos automatizados de gestión de tráfico. Adicionalmente, se han propuesto esquemas de gestión de recursos en entornos cloud-edge que incorporan mecanismos de confianza para mejorar la seguridad y la toma de decisiones en redes distribuidas (Saini & Gupta, 2023; Liu et al., 2021).

En el ámbito agroalimentario, estas tecnologías permiten implementar sistemas avanzados de monitoreo y trazabilidad. Los sensores IoT pueden registrar variables como temperatura, humedad, ubicación o condiciones de almacenamiento, generando información que permite mejorar la transparencia y control en la cadena de suministro alimentaria (Rahman et al., 2022). La capacidad de SDN para gestionar dinámicamente estos flujos de información contribuye a mejorar la eficiencia operativa, la toma de decisiones y la confiabilidad de los sistemas de trazabilidad en entornos productivos.

A pesar de sus ventajas, las arquitecturas SDN-IoT presentan diversos desafíos de seguridad derivados de su naturaleza distribuida y altamente conectada. La centralización del plano de control, aunque mejora la gestión de la red, también introduce nuevos vectores de ataque que pueden comprometer la disponibilidad y confiabilidad del sistema. En este contexto, el diseño arquitectónico debe considerar estrategias que permitan distribuir funciones de red y optimizar el uso de recursos, como es el caso de enfoques basados en virtualización de funciones y computación en el borde, los cuales contribuyen a mejorar la resiliencia del sistema (Alsmadi & Xu, 2019; Chen et al., 2022).

Uno de los principales riesgos en estas arquitecturas corresponde a los ataques de denegación de servicio distribuido (DDoS), los cuales buscan saturar los recursos de red mediante la generación masiva de tráfico malicioso. En entornos IoT, donde miles de dispositivos pueden estar conectados simultáneamente, estos ataques pueden afectar tanto a los dispositivos finales como al controlador SDN encargado de gestionar la red, evidenciando la necesidad de mecanismos eficientes de mitigación y control del tráfico (Singh et al., 2019).

Asimismo, la comunicación constante entre sensores, dispositivos y servidores introduce riesgos relacionados con la interceptación y manipulación de datos. En sistemas de trazabilidad agroalimentaria, la integridad de la información es crítica, ya que los datos recopilados permiten verificar el origen, estado y condiciones de los productos a lo largo de la cadena de suministro. En este sentido, la adopción de arquitecturas híbridas que optimicen la ubicación de funciones de red y el procesamiento de datos puede contribuir a mejorar la eficiencia y reducir vulnerabilidades en la transmisión de información (Rahman et al., 2022; Li et al., 2024).

Otro aspecto relevante en las arquitecturas basadas en IoT es la protección de la privacidad y confidencialidad de la información generada por los dispositivos conectados. En entornos SDN-IoT, resulta necesario implementar mecanismos que garanticen la autenticación segura de los dispositivos, la protección de las comunicaciones y el control de acceso a los recursos de la red. En este contexto, Rao et al. (2024) proponen esquemas de comunicación que preservan la privacidad mediante el uso de mecanismos de autenticación y transmisión segura de datos, lo cual contribuye a reducir el riesgo de exposición de información sensible y a fortalecer la seguridad en redes IoT gestionadas mediante SDN.

En entornos industriales y agroalimentarios, la seguridad también se relaciona con la continuidad operativa de las infraestructuras. Un ataque exitoso puede afectar sistemas de

monitoreo, automatización o logística, generando pérdidas económicas y riesgos en la calidad de los productos. Por esta razón, diversos estudios proponen integrar mecanismos avanzados de detección de intrusiones y monitoreo del tráfico en redes SDN-IoT, así como estrategias de optimización de recursos que permitan mantener la operación del sistema incluso en condiciones adversas (Rahman et al., 2021).

La investigación reciente ha propuesto diversas tecnologías orientadas a fortalecer la seguridad y resiliencia de las arquitecturas SDN-IoT. Entre estas soluciones destacan los mecanismos de control de acceso seguro, las arquitecturas basadas en blockchain y las técnicas de orquestación de red orientadas a optimizar el uso de recursos y mejorar la respuesta ante condiciones dinámicas del sistema.

Los protocolos de control de acceso desempeñan un papel fundamental en la protección de redes IoT, ya que permiten autenticar dispositivos y restringir el acceso a los recursos de la red. En entornos con un gran número de dispositivos conectados, es necesario implementar mecanismos escalables y seguros. En este contexto, Bhatia et al. (2024) proponen un protocolo de control de acceso diseñado para entornos IoT que gestiona de manera eficiente la autenticación y autorización de dispositivos, reduciendo el riesgo de intrusiones y accesos no autorizados.

Por otra parte, la tecnología blockchain ha sido ampliamente explorada como mecanismo para garantizar la integridad y trazabilidad de la información generada por dispositivos IoT. En sistemas agroalimentarios, permite almacenar registros inmutables de los eventos ocurridos a lo largo de la cadena de suministro, facilitando la verificación de la información y mejorando la transparencia del sistema (Alam et al., 2023; Rahman et al., 2021).

En el ámbito de la gestión de redes, diversas investigaciones han abordado el uso de técnicas de aprendizaje automático y aprendizaje por refuerzo profundo para optimizar la asignación de recursos y la orquestación de funciones de red en arquitecturas SDN-IoT. Estas soluciones permiten mejorar la eficiencia operativa y la resiliencia del sistema ante fallos o congestión de tráfico (Zhou et al., 2024; Zhang et al., 2020).

De manera complementaria, se han desarrollado enfoques orientados a la optimización de la ubicación de funciones de red virtualizadas y a la mejora de la tolerancia a fallos en infraestructuras complejas. En particular, algunos trabajos proponen modelos basados en aprendizaje profundo que permiten adaptar dinámicamente la red a condiciones cambiantes, incrementando la robustez y continuidad del servicio (Wang et al., 2021; Zhao et al., 2023).

En conjunto, estas tecnologías proporcionan herramientas fundamentales para construir arquitecturas SDN-IoT más seguras, resilientes y eficientes. A partir de este análisis, se elaboró una matriz comparativa que evalúa variables como arquitectura, mecanismos de seguridad, escalabilidad, latencia y viabilidad económica, permitiendo identificar patrones, ventajas y limitaciones en las soluciones existentes como base para la definición del marco SDN propuesto.

Tabla 1

Matriz comparativa de arquitecturas y mecanismos de seguridad en redes SDN-IoT

Trabajo	Arquitectura	Seguridad	Gestión de tráfico	Escalabilidad	Latencia	Bajo costo	Aplicación
Alsmadi & Xu (2019)	SND	Identificación de vulnerabilidades y mecanismos de protección	Media	Alta	Media	Media	Redes definidas por software
Awasthi & Sharma (2024)	SDN + IA	Detección inteligente de anomalías	Alta	Alta	Baja	Media	Industrial IoT
Fu et al. (2020)	SDN + NFV	Seguridad mediante orquestación inteligente	Alta	Alta	Baja	Media	Redes IoT virtualizadas
Jiang et al. (2022)	Cloud-Edge SDN	Gestión confiable de recursos	Alta	Alta	Baja	Media	IoT distribuido
Liu et al. (2021)	SDN + MEC	Orquestación segura de servicios	Alta	Alta	Baja	Media	IoT con edge computing
Rahman et al. (2021)	SDN	Mitigación de ataques DDoS	Alta	Media	Media	Media	Redes industriales
Rahman et al. (2022)	IoT + Blockchain	Integridad de datos y trazabilidad	Media	Alta	Media	Media	Cadena agroalimentaria
Rossi et al. (2025)	IoT	Trazabilidad y monitoreo logístico	Media	Media	Media	Media	Cadena de suministro alimentaria
Saini & Gupta (2023)	SDN-IoT	Seguridad básica de red	Media	Alta	Baja	Alta	Automatización IoT de bajo costo

Singh et al. (2019)	SDN-IoT	Detección y mitigación DDoS	Alta	Media	Media	Media	Redes IoT
Wang et al. (2021)	SDN + DRL	Tolerancia a fallos	Alta	Alta	Baja	Media	Redes virtualizadas
Zhang et al. (2023)	MEC + SDN	Optimización de recursos	Alta	Alta	Baja	Media	Edge computing
Zhou et al. (2024)	SDN + DRL	Seguridad basada en aprendizaje profundo	Alta	Alta	Baja	Media	IoT inteligente
Amin et al. (2023)	SDN-IoT	Arquitectura segura	Alta	Alta	Media	Media	Infraestructura IoT
Alam et al. (2023)	SDN + Blockchain	Seguridad e integridad de datos	Alta	Alta	Media	Media	Industrial IoT
Bhatia & Singh (2024)	IoT	Control de acceso seguro	Alta	Alta	Media	Media	Dispositivos IoT
Li et al. (2024)	SDN	Revisión de arquitecturas y seguridad	Media	Alta	Media	Media	Redes definidas por software
Rao & Das (2024)	SDN-IoT	Comunicación preservando privacidad	Alta	Media	Media	Media	Smart homes

Nota. Esta tabla muestra el análisis comparativo de todos los trabajos de la bibliografía. *Fuente*. Autoria propia.

Dentro de los estudios analizados, se identifican aportes relevantes en gestión del tráfico, seguridad y resiliencia de red. Los enfoques basados en aprendizaje por refuerzo profundo (DRL), como los propuestos por Fu et al., Wang et al. y Zhou et al., destacan por su alta capacidad de optimización de recursos y escalabilidad. En paralelo, los trabajos orientados a la mitigación de ataques DDoS, como los de Rahman et al. y Singh et al., evidencian una fuerte

orientación hacia la seguridad, al permitir la detección de anomalías y la aplicación de políticas de control desde el controlador SDN. Asimismo, propuestas como la de Saini y Gupta resaltan en términos de viabilidad económica, al basarse en hardware estándar y soluciones de código abierto.

En conjunto, la matriz comparativa evidencia que las investigaciones tienden a abordar de forma aislada aspectos como seguridad, optimización o trazabilidad, lo que revela una oportunidad en la integración de estos elementos dentro de un marco arquitectónico unificado. Este análisis fundamenta los lineamientos del modelo SDN propuesto en el Capítulo 4, orientado a combinar gestión eficiente del tráfico, mecanismos de seguridad y bajo costo para su aplicación en sistemas de trazabilidad IoT en el sector agroalimentario.

Objetivo Específico 2

Para contextualizar el diseño del marco arquitectónico propuesto, se plantea una representación conceptual basada en tres capas fundamentales: capa IoT, capa de red SDN y capa de aplicación. Esta estructura se organiza en forma de triángulo con el fin de ilustrar la relación jerárquica y funcional entre los componentes del sistema, donde la generación de datos se realiza en la base, su gestión y control en el nivel intermedio, y su procesamiento y visualización en el nivel superior. Este enfoque permite comprender de manera integral cómo interactúan los diferentes elementos de la arquitectura para garantizar la recolección, transmisión, procesamiento y aprovechamiento de la información en sistemas de trazabilidad agroalimentaria.

Figura 1

Estructura por capas de marco arquitectónico

Nota. Marco arquitectónico dividido por capas. *Fuente*. Autoria propia.

La arquitectura propuesta se basa en la integración de redes definidas por software (SDN) con entornos de Internet de las Cosas (IoT), con el objetivo de mejorar la gestión del tráfico, fortalecer la seguridad y garantizar la eficiencia operativa en sistemas de trazabilidad agroalimentaria. Este enfoque se estructura en tres capas principales: capa de dispositivos IoT, capa de red SDN y capa de aplicación, permitiendo una separación funcional que facilita la administración y escalabilidad del sistema (Kreutz et al., 2015; Saini & Gupta, 2023).

En la capa de dispositivos IoT, se encuentran los sensores encargados de recolectar información relacionada con variables críticas como temperatura, humedad y ubicación de los

productos a lo largo de la cadena de suministro. Estos dispositivos generan flujos de datos que son transmitidos hacia la infraestructura de red, constituyendo la base del sistema de trazabilidad. La naturaleza distribuida y heterogénea de estos dispositivos requiere mecanismos eficientes de gestión y control que permitan garantizar la confiabilidad de la información generada (Rahman et al., 2022; Rossi et al., 2025).

La capa de red SDN constituye el núcleo de la arquitectura y está compuesta por dispositivos de conmutación compatibles con OpenFlow y un controlador SDN centralizado. El controlador actúa como entidad lógica encargada de gestionar el comportamiento de la red mediante la instalación de reglas de flujo en los dispositivos, lo que permite implementar políticas dinámicas de encaminamiento, priorización del tráfico y segmentación de la red. Este enfoque facilita la optimización del tráfico generado por los dispositivos IoT, permitiendo diferenciar flujos críticos y mejorar la eficiencia en la transmisión de datos (Kreutz et al., 2015; Zhou et al., 2024).

En esta misma capa se integran mecanismos de control de acceso, los cuales permiten autenticar los dispositivos y restringir el acceso a los recursos de red mediante políticas definidas desde el controlador. Estos mecanismos se basan en la identificación de dispositivos y en la aplicación de reglas que limitan el tráfico únicamente a nodos autorizados, reduciendo el riesgo de accesos no autorizados y fortaleciendo la seguridad del sistema (Bhatia et al., 2024; Singh et al., 2019).

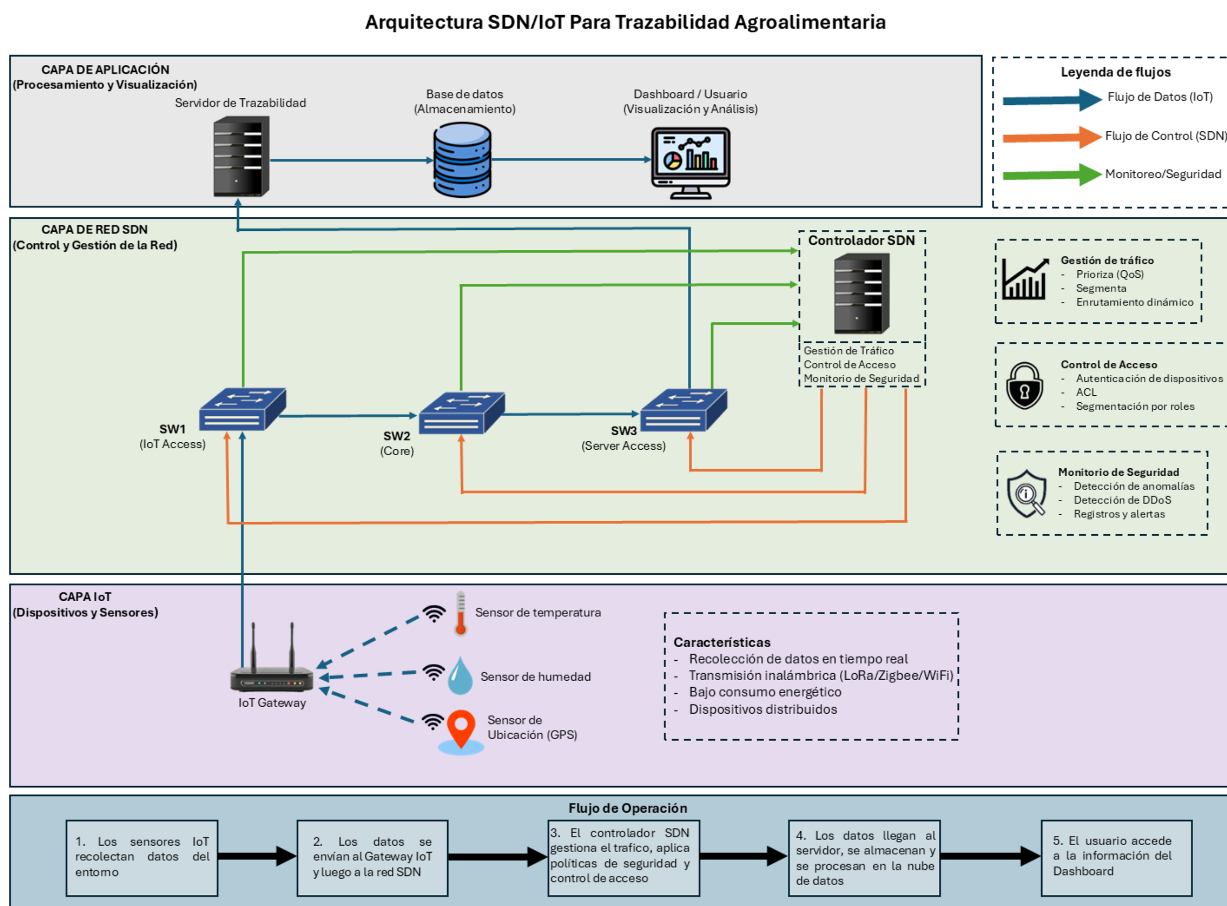
Adicionalmente, la arquitectura incorpora un módulo de monitoreo de seguridad, el cual permite supervisar el comportamiento del tráfico en la red y detectar posibles anomalías, como patrones asociados a ataques de denegación de servicio. A partir de esta información, el controlador SDN puede aplicar medidas de mitigación de forma dinámica, como el bloqueo de

flujos sospechosos o la redistribución del tráfico, mejorando la resiliencia del sistema frente a amenazas externas (Singh et al., 2019; Rahman et al., 2021).

Finalmente, en la capa de aplicación se ubican los sistemas encargados del procesamiento, almacenamiento y visualización de los datos de trazabilidad. Estos sistemas permiten analizar la información generada por los sensores y apoyar la toma de decisiones en los procesos logísticos y productivos. El flujo de datos en la arquitectura sigue un esquema ascendente, desde los dispositivos IoT hacia las aplicaciones, mientras que las políticas de control y gestión se aplican de manera descendente desde el controlador SDN hacia los dispositivos de red, estableciendo un modelo centralizado y programable que optimiza el funcionamiento general del sistema. La siguiente figura muestra el diseño de la arquitectura propuesta.

Figura 2

Arquitectura SDN/IoT



Nota. Arquitectura SDN/IoT. *Fuente*. Autoria propia.

En la arquitectura propuesta se puede observar una clara separación funcional entre el plano de datos, el plano de control y el plano de monitoreo, lo cual responde al paradigma de redes definidas por software (SDN), donde la gestión de la red se centraliza y se desacopla del reenvío de paquetes. Esta organización permite optimizar la administración del tráfico, mejorar la seguridad y facilitar la escalabilidad del sistema, especialmente en entornos IoT orientados a la trazabilidad agroalimentaria.

En primer lugar, el flujo de datos IoT (representado en azul) sigue una trayectoria

unidireccional desde los dispositivos de adquisición hasta los sistemas de almacenamiento y visualización. Los sensores generan información relacionada con variables del entorno, como temperatura, humedad o ubicación, la cual es recolectada por el Gateway IoT. Este dispositivo actúa como punto de agregación y adaptación de protocolos, permitiendo encapsular y transmitir los datos hacia la red definida por software. Posteriormente, la información atraviesa los switches SDN (SW1, SW2 y SW3), los cuales se encargan exclusivamente del reenvío de paquetes según las reglas previamente instaladas. Finalmente, los datos llegan al servidor de trazabilidad, donde son procesados y almacenados en la base de datos, para luego ser visualizados en el dashboard.

En segundo lugar, el flujo de control (representado en naranja) se origina en el controlador SDN y se dirige hacia los switches de la red. Aunque en una implementación real el canal de control es bidireccional, en la arquitectura se representa de forma unidireccional para enfatizar el rol central del controlador en la toma de decisiones. El controlador, implementado sobre un servidor Linux, incorpora módulos lógicos de gestión de tráfico y control de acceso, los cuales permiten definir políticas de calidad de servicio, priorización de flujos y restricciones de comunicación entre dispositivos. Estas decisiones se materializan mediante la instalación de reglas de flujo en los switches, que determinan cómo se deben tratar los paquetes en función de criterios previamente definidos. De esta forma, los switches operan como elementos de reenvío programables, ejecutando las políticas establecidas por el controlador sin necesidad de tomar decisiones autónomas complejas.

Por otro lado, el flujo de monitoreo y seguridad (representado en verde) cumple la función de proporcionar retroalimentación sobre el estado de la red. Este flujo se origina en los switches (SW1, SW2 y SW3), los cuales generan estadísticas de tráfico, registros de eventos y

métricas operativas que son enviadas hacia el módulo de monitoreo de seguridad dentro del servidor del controlador SDN y se encarga de analizar la información recibida para detectar comportamientos anómalos, posibles ataques o condiciones de congestión. Posteriormente, los resultados del análisis son enviados al controlador, donde se toman decisiones correctivas que se aplican nuevamente a través del flujo de control. Aunque en la práctica pueden existir intercambios bidireccionales, en el modelo se representa este flujo en sentido ascendente para resaltar su función principal de recolección de información y soporte a la toma de decisiones.

En cuanto a la composición de la arquitectura, se identifican claramente tres tipos de componentes: dispositivos de campo, infraestructura de red y sistemas de gestión. Los sensores y el Gateway IoT conforman la capa de adquisición de datos; los switches SDN constituyen la infraestructura de transporte; y el controlador SDN, junto con el servidor de trazabilidad, la base de datos y el dashboard, integran la capa de procesamiento y gestión. Adicionalmente, los módulos de gestión de tráfico, control de acceso y monitoreo de seguridad son funciones lógicas implementadas dentro del controlador, lo que permite una gestión centralizada y coherente de toda la red.

Objetivo Específico 3

La validación experimental propuesta se centra en analizar el comportamiento de una arquitectura de red basada en redes definidas por software (SDN) aplicada a un entorno de Internet de las Cosas (IoT), mediante el uso de escenarios controlados en un entorno de simulación. El objetivo principal es examinar el desempeño de la red frente a diferentes condiciones de tráfico, permitiendo identificar mejoras en términos de gestión, eficiencia y seguridad.

En este contexto, la validación se estructura en tres componentes fundamentales. En primer lugar, la capa de red basada en SDN, encargada de centralizar el control del tráfico y aplicar políticas dinámas mediante un controlador lógico. En segundo lugar, un entorno IoT simulado, en el cual se generan flujos de datos representativos del comportamiento de sensores, permitiendo modelar condiciones reales de operación. Finalmente, se incorpora una capa de aplicación básica orientada a la visualización y almacenamiento de los datos, con el fin de evidenciar el flujo de información dentro del sistema.

De esta manera, la validación permite comparar el desempeño de la red bajo distintos escenarios, incluyendo condiciones con y sin la intervención del controlador SDN, así como situaciones de carga elevada y eventos de seguridad. Esto facilita analizar el impacto del enfoque SDN en aspectos como latencia, pérdida de paquetes, uso del ancho de banda y capacidad de respuesta del sistema, bajo un entorno controlado y reproducible.

El entorno de simulación fue diseñado utilizando herramientas de virtualización y emulación de redes que permiten reproducir de manera controlada el comportamiento de una infraestructura IoT basada en SDN. Para la implementación de la topología y la interconexión de

los dispositivos se empleó GNS3, el cual permite integrar máquinas virtuales y simular enlaces de red de forma flexible, facilitando la construcción de escenarios complejos.

Como plataforma de virtualización se utilizó VirtualBox, donde se desplegaron múltiples máquinas virtuales con sistema operativo Ubuntu 22.04, cada una cumpliendo un rol específico dentro de la arquitectura. Estas máquinas representan los diferentes componentes del sistema, incluyendo dispositivos IoT, switches virtuales, controlador SDN y servidor de trazabilidad.

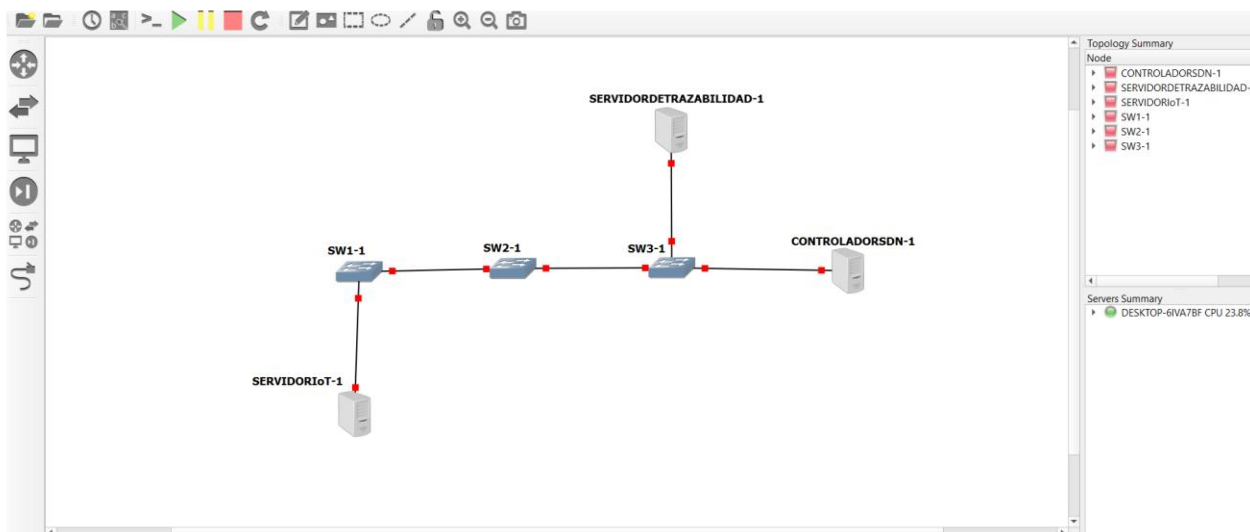
Para el análisis del tráfico de red se utilizó Wireshark, herramienta que permite capturar y examinar paquetes en tiempo real, facilitando la evaluación de métricas como latencia, pérdida de paquetes y comportamiento del tráfico. Adicionalmente, se empleó iperf3 para la generación de tráfico controlado y la medición de throughput, lo que permite simular condiciones de carga en la red.

Finalmente, se utilizó Python como herramienta para la generación de tráfico IoT simulado, permitiendo enviar datos de forma periódica desde los nodos sensores hacia el servidor de trazabilidad. Esto permite reproducir un flujo continuo de información similar al de un entorno IoT real, facilitando la evaluación del comportamiento de la red bajo diferentes condiciones operativas.

La siguiente figura muestra la topología implementada en GNS3 de la arquitectura.

Figura 3

Simulación en GNS3 de la arquitectura



Nota. Simulación con dispositivos en GNS3. *Fuente*. Autoria propia.

La arquitectura implementada en GNS3 representa un entorno simplificado de una red IoT basada en SDN, estructurada en diferentes componentes que permiten simular tanto la generación de datos como su transporte, control y almacenamiento. Esta arquitectura se diseñó con el propósito de evaluar el comportamiento del tráfico dentro de la red y analizar el impacto del controlador SDN en su gestión.

En la capa de generación de datos se encuentran los sensores simulados, representados por una máquina virtual que actúa como servidor IoT. Este nodo genera tráfico de manera periódica utilizando scripts en Python, simulando el envío de datos desde dispositivos IoT hacia la red.

El gateway está representado por el primer switch (SW1), el cual actúa como punto de entrada del tráfico IoT hacia la infraestructura de red. A partir de este punto, los datos atraviesan una serie de switches virtuales (SW1, SW2 y SW3), implementados mediante Open vSwitch, que

permiten el encaminamiento del tráfico y la aplicación de reglas definidas por el controlador SDN.

El controlador SDN se encuentra conectado al switch principal (SW3) y es el encargado de gestionar la red de forma centralizada. A través de este componente se implementan políticas de control de tráfico, priorización de flujos y mecanismos básicos de seguridad, mediante la instalación de reglas de flujo en los switches.

Finalmente, el servidor de trazabilidad representa la capa de aplicación, donde se reciben, almacenan y procesan los datos provenientes de los dispositivos IoT. Este servidor cuenta con una base de datos, una API básica que permite gestionar la información y el dashboard que constituye la interfaz de visualización del sistema, permitiendo observar el flujo de datos.

Para el criterio de diseño de la red se empleó la Red base 10.0.0.0/24.

La siguiente tabla muestra el subnetting de la red.

Tabla 2

Subnetting de la red

Dispositivo	Interfaz en GNS3	IP	Conecta a
Servidor IoT	Eth1	10.0.0.10/24	SW1
SW1 (bridge br0)	br0	10.0.0.1/24	SW2/Servidor
SW2 (bridge br0)	br0	10.0.0.2/24	SW1/SW3
SW3 (bridge br0)	br0	10.0.0.3/24	SW2/ Controlador SDN/
Controlador SDN	Eth1	10.0.0.20/24	Servidor de trazabilidad
Servidor de trazabilidad	Eth1	10.0.0.30/24	SW3 SW3

Nota. Esta tabla muestra el subnetting de la red. *Fuente*. Autoria propia.

Para la configuración hay que tener en cuenta que no se realizará un Routing tradicional, debido a que OVS funciona con flows (OpenFlow) que es el protocolo estandarizado de código abierto para SDN, sin embargo, si se requiere realizar la configuración de las IPs, la conectividad base y las rutas estáticas en los servidores y en el controlador.

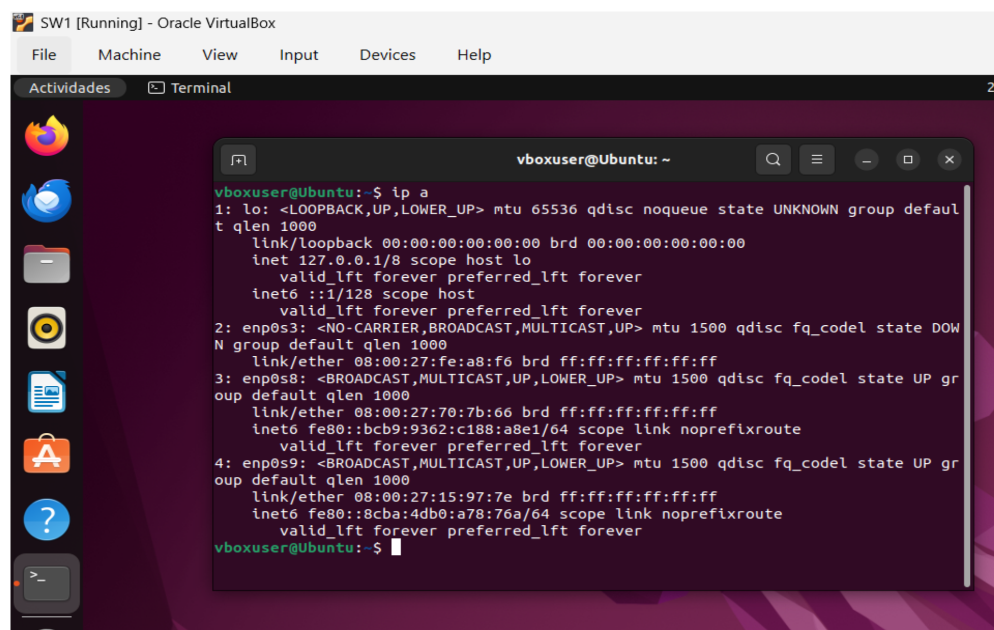
Aunque se configuran rutas estáticas básicas en los nodos finales para garantizar conectividad inicial, el encaminamiento del tráfico en la red no depende de protocolos tradicionales, sino de reglas dinámicas definidas por el controlador SDN mediante OpenFlow.

Se iniciará con la configuración de las interfaces para SW1, SW2 y SW3.

Para SW1, se realiza la verificación de las interfaces.

Figura 4

Interfaces SW1



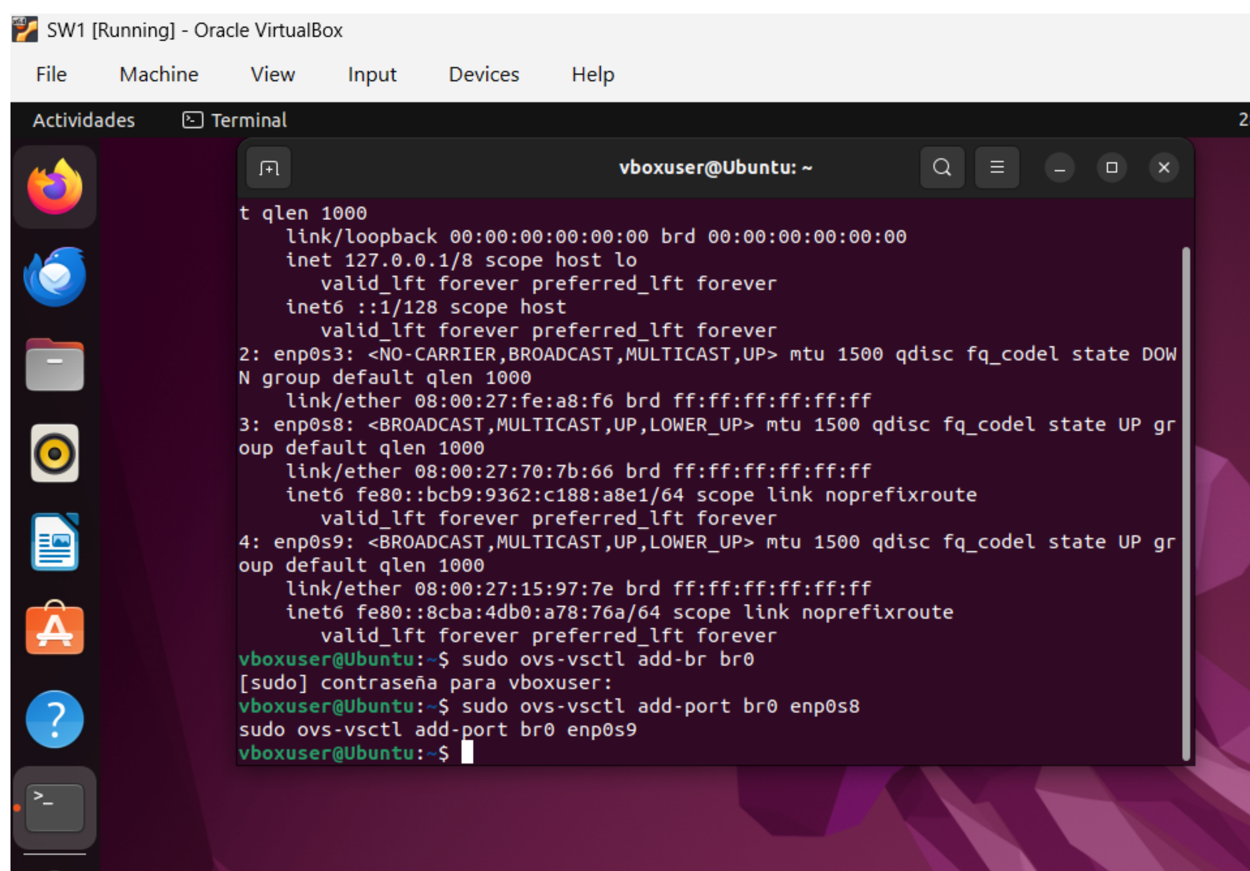
Nota. Muestra de Interfaces SW1. *Fuente*. Autoria propia.

Se puede ver que en Linux se emplea una interfaz con un nombre distinto, para este caso el equivalente a Ethernet0 es enp0s3 que está configurado en NAT y no se usará, mientras que Ethernet1 es enp0s8 y Ethernet2 es enp0s9 que son las que usaremos.

A continuación, se procederá con la creación del Bridge y se agregarán las interfaces de Linux.

Figura 5

Creación de Bridge con las interfaces



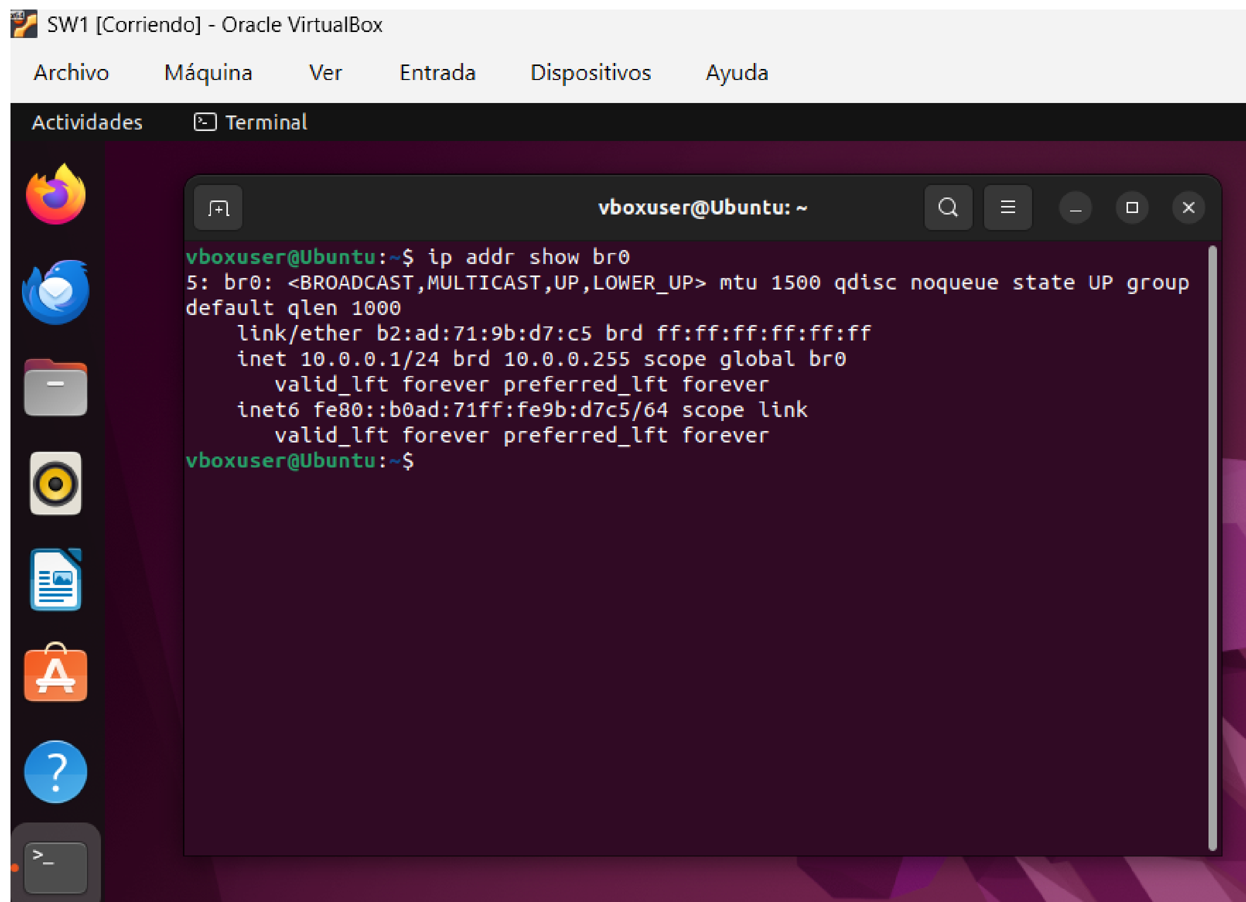
Nota. Creación de Bridge en SW1. *Fuente*. Autoria propia.

Posteriormente se levantan las interfaces y se configura la IP de gestión 10.0.0.1/24, en la siguiente imagen se realiza la validación de la configuración, donde se puede ver el Bridge

creado br0 con la IP de gestión y las interfaces agregadas Port enp0s8 y Port enp0s9, que conectarán a Servidor IoT y al SW2 respectivamente.

Figura 6

Validación de configuración SW1



Nota. Validación de Bridge en SW1. *Fuente*. Autoria propia.

Ahora se realizará el mismo procedimiento con SW2, primero mirando la asignación actual de las interfaces.

Figura 7

Interfaces SW2

SW2 [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Actividades

vboxuser@Ubuntu: ~

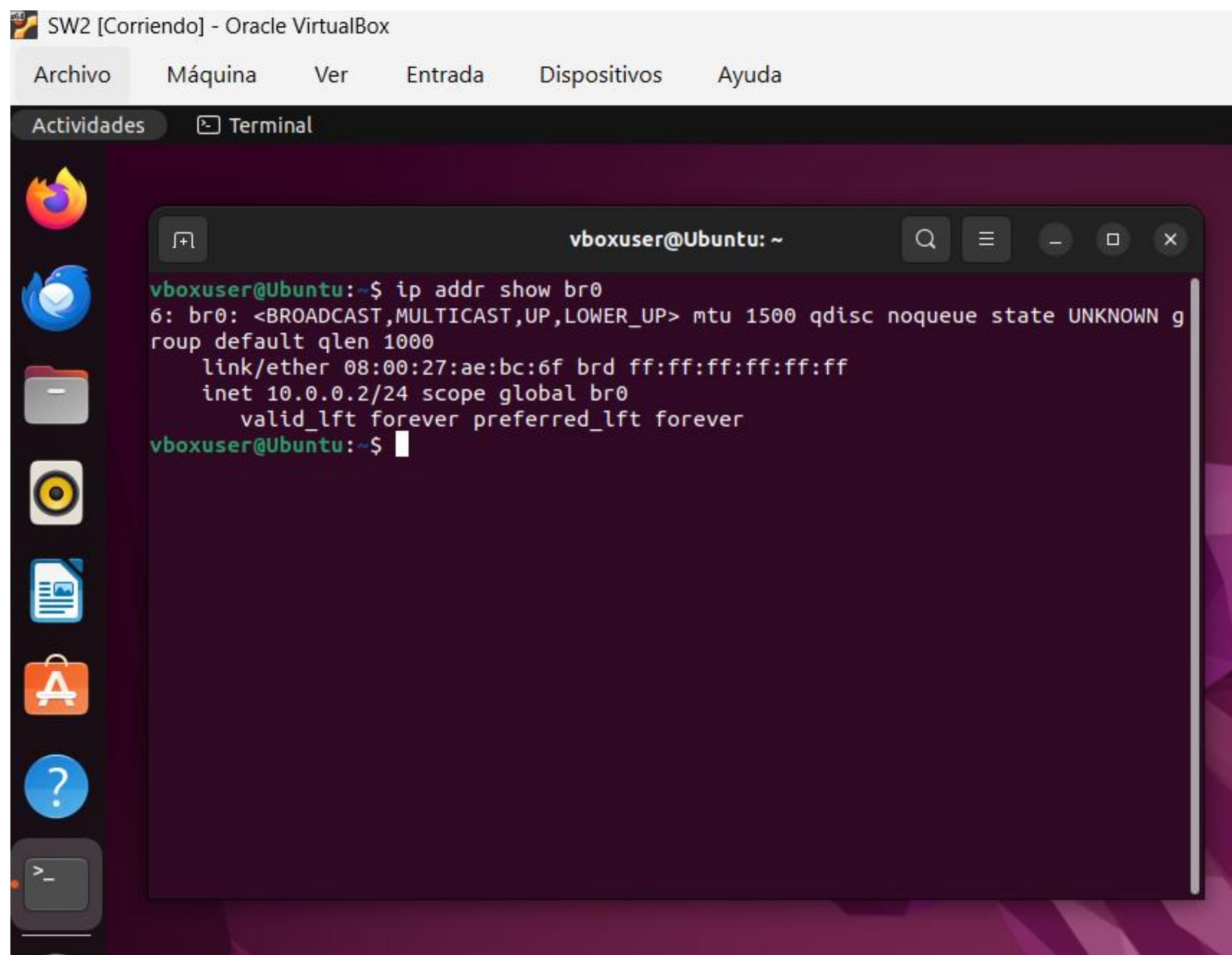
```
vboxuser@Ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOW
N group default qlen 1000
    link/ether 08:00:27:fe:a8:f6 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:e6:1e:60 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ece:afbf:95b4:43b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:ae:bc:6f brd ff:ff:ff:ff:ff:ff
    inet6 fe80::b2db:ede1:250d:c1bd/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
vboxuser@Ubuntu:~$
```

Nota. Muestra de interfaces en SW2. *Fuente*. Autoria propia.

Se puede ver el mismo escenario donde enp0s8 será la conexión hacia SW1 y enp0s9 será la conexión hacia SW3, a continuación, se mostrará la validación de la configuración del Brige la IP de gestión 10.0.0.2/24.

Figura 8

Validación de configuración SW2



Nota. Validación de Bridge en SW2. *Fuente*. Autoria propia.

Por último, en esta parte se seguirá con la configuración de SW3, se validarán las interfaces, en la siguiente imagen se pueden observar las tres interfaces con las que funcionará SW3

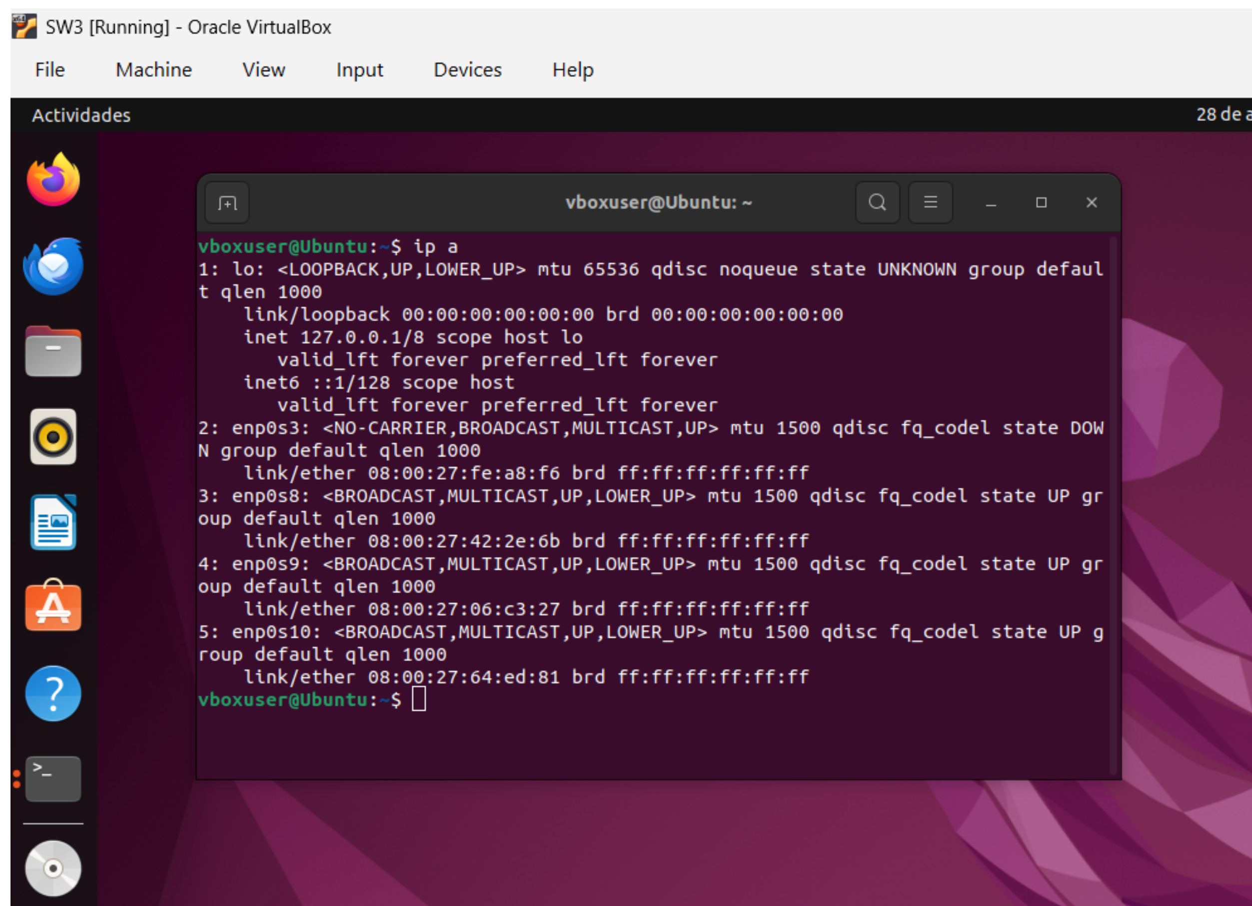
Enp0s8 va hacia SW2.

Enp0s9 va hacia Controlador.

Enp0s10 va hacia Servidor.

Figura 9

Interfaces SW3

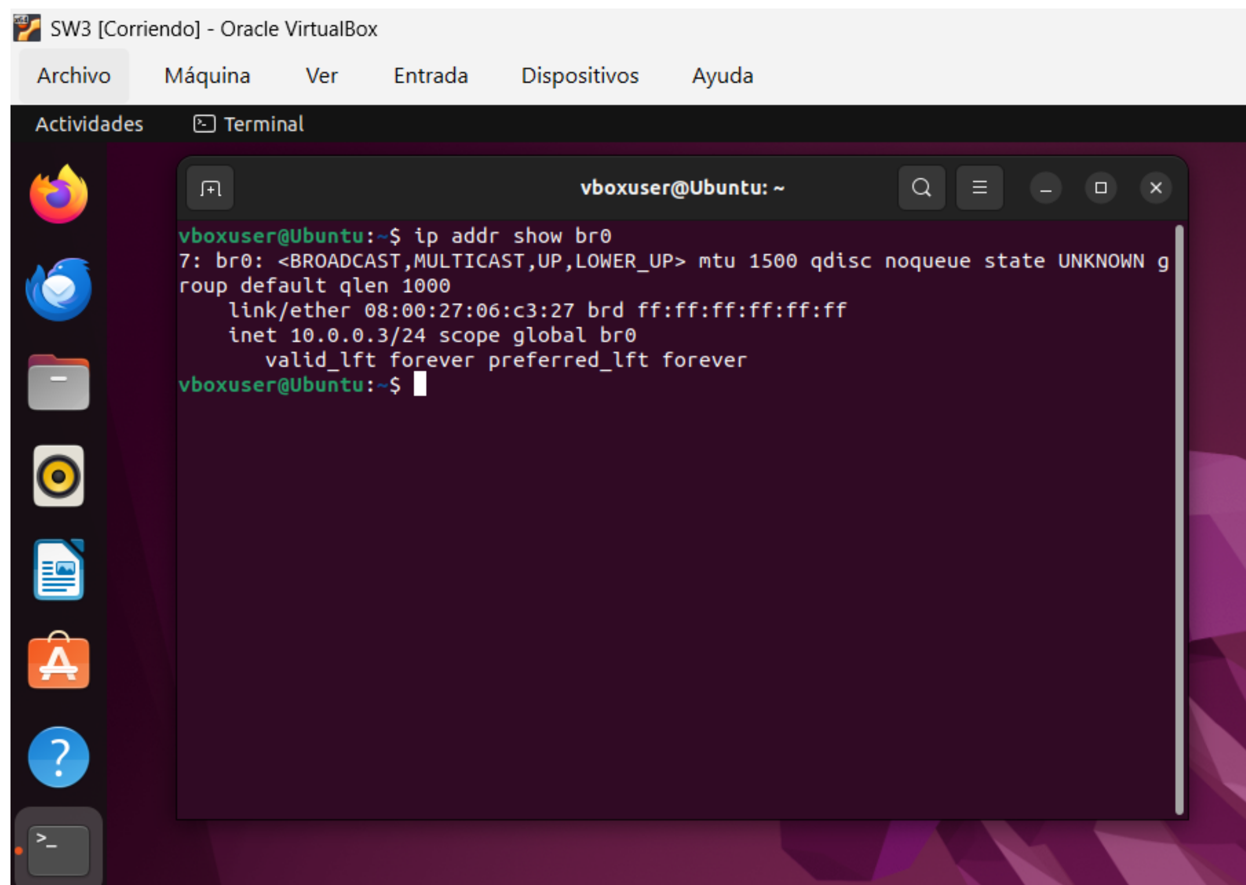


Nota. Muestra de interfaces en SW3. *Fuente*. Autoria propia.

Se realiza la misma configuración y se valida el Bridge br0 creado, las tres interfaces conectadas, la IP de gestión 10.0.0.3/24 y el bridge en UP.

Figura 10

Validación de configuración SW3



Nota. Validación de configuración en SW3. *Fuente*. Autoria propia.

Los switches fueron configurados utilizando Open vSwitch mediante la creación de un bridge virtual (br0), al cual se asociaron las interfaces físicas correspondientes a cada enlace. Este enfoque permite desacoplar el plano de control del plano de datos, facilitando la posterior gestión mediante el controlador SDN.

Hay que tener en cuenta que para estos Switches no se ha asignado IPs a cada interfaz física, sino que se asignó únicamente al bridge br0, debido a que:

Open vSwitch funciona como switch de capa 2.

El forwarding real lo controla el controlador SDN mediante flows.

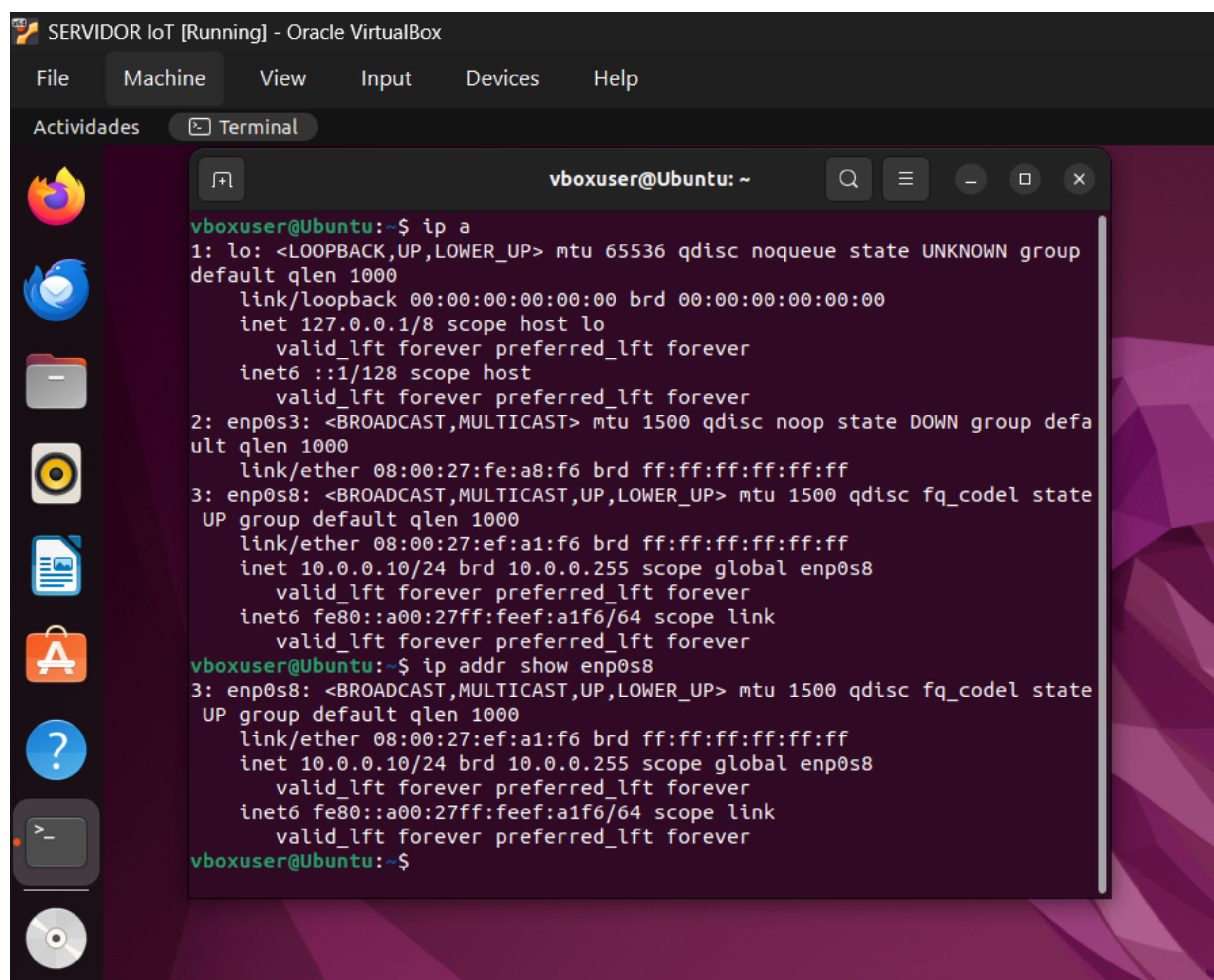
La IP en br0 de gestión que se configuró en cada Switch solo sirve para pruebas de conectividad y gestión básica.

Ahora se realizará la configuración en el Servidor IoT, de igual forma se validan las interfaces, donde únicamente se utilizará el enp0s8 como interfaz que va hacia SW1.

La siguiente imagen muestra la asignación de la IP 10.0.0.10/24 y donde se levanta la interfaz.

Figura 11

Configuración interfaz Servidor IoT

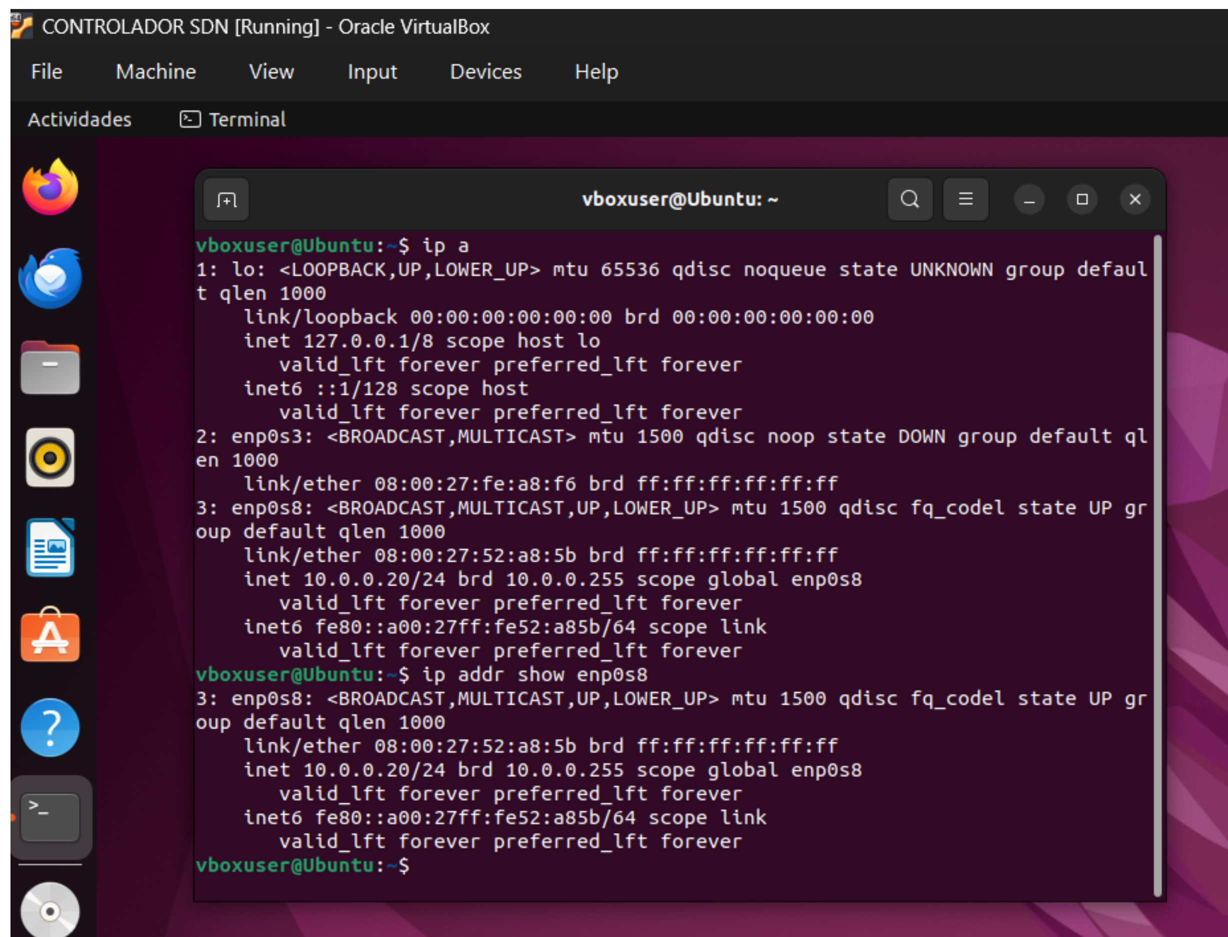


Nota. Configuración de interfaces en servidor IoT. *Fuente*. Autoria propia.

A continuación, se realiza la misma configuración del controlador SND, aquí se puede observar que se configura la dirección IP 10.0.0.20/24 en la interfaz enp0s8 del controlador SDN.

Figura 12

Configuración interfaz Controlador SDN

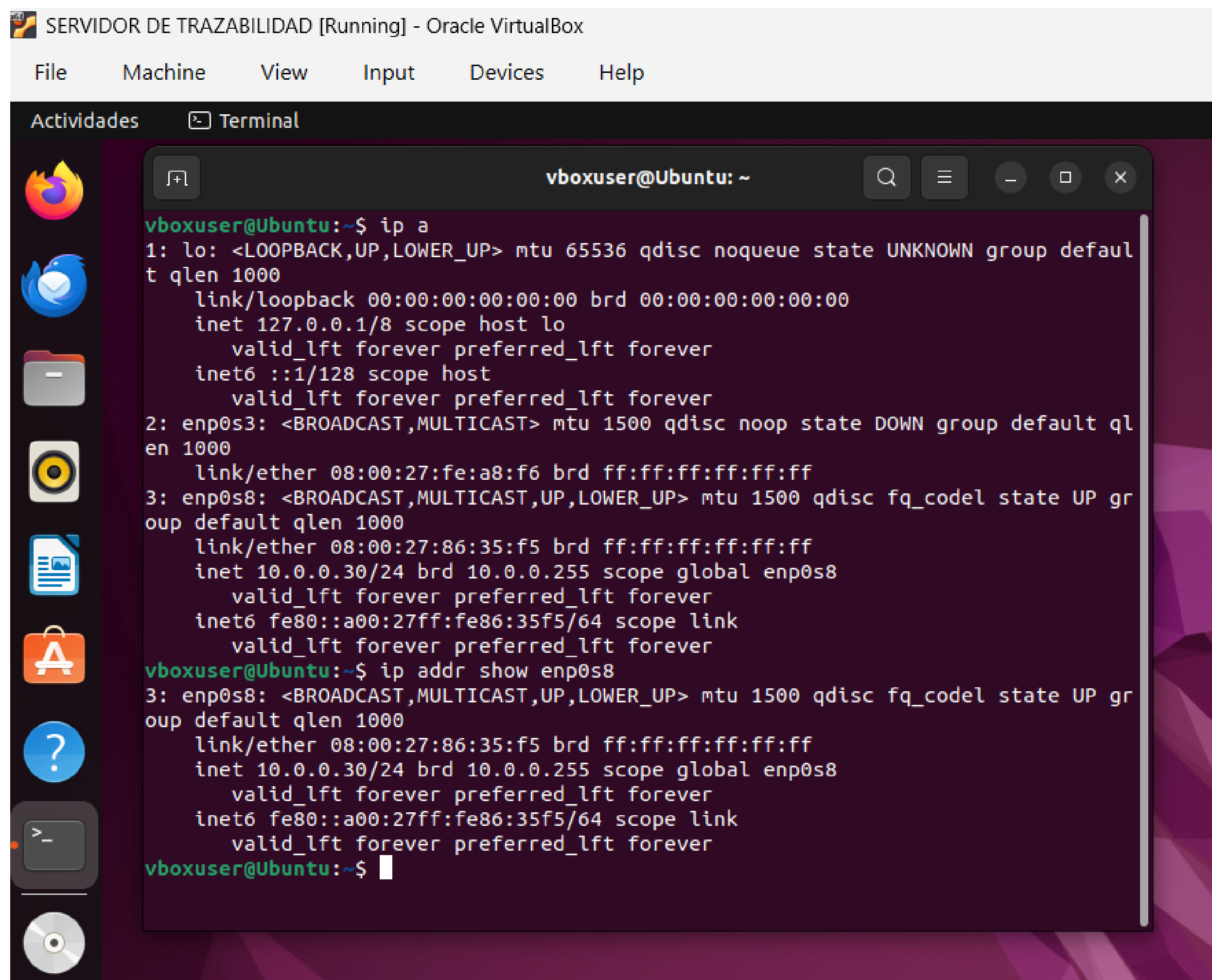


Nota. Configuración de interfaces en servidor SDN. *Fuente*. Autoria propia.

Por último, se hace el mismo procedimiento en el servidor de trazabilidad, se configura la dirección IP 10.0.0.30/24 en la interfaz enp0s8.

Figura 13

Configuración interfaz Servidor de trazabilidad



Nota. Configuración de interfaces en servidor de trazabilidad. *Fuente*. Autoria propia.

Además de esto, en los hosts que tenemos como servidores (IoT, Controlador y Trazabilidad), se agregó lo siguiente:

10.0.0.0/24 dev enp0s8 scope link src X.X.X.X

Esto es una ruta directamente conectada, que dirá: Todo lo que esté en 10.0.0.X lo envío. directamente por mi interfaz.

Con esto, procedemos a realizar las pruebas de conexión.

Figura 14

Ping de Servidor IoT a SW1

SERVIDOR IoT [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Actividades Terminal
vboxuser@Ubuntu: ~

```
vboxuser@Ubuntu:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1.46 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.784 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=1.17 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.800 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.950 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=1.99 ms
^C
--- 10.0.0.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5069ms
rtt min/avg/max/mdev = 0.784/1.192/1.986/0.424 ms
vboxuser@Ubuntu:~$
```

Nota. Prueba de Ping entre servidores. *Fuente*. Autoria propia.

En este caso, el Servidor IoT revisa su tabla:

10.0.0.0/24 dev enp0s8.

Dice: 10.0.0.1 está en mi red local.

Aquí no se está utilizando Gateway, no se enruta, únicamente se hace ARP:

“¿Quién tiene 10.0.0.1?”.

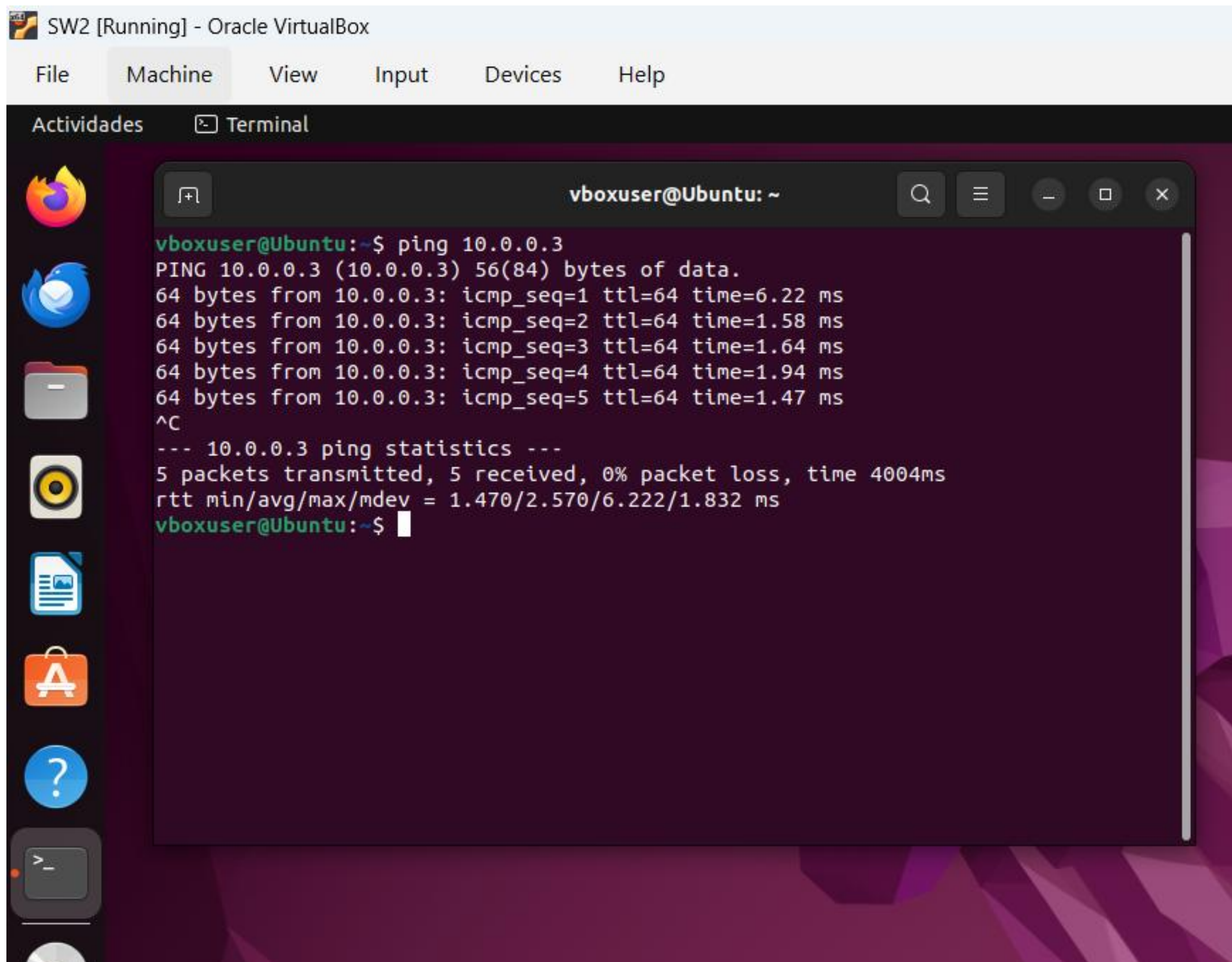
SW1 responde con su MAC.

OVS reenvía el frame.

Para el caso de SW2 y SW3 es lo mismo.

Figura 15

Ping de SW2 a SW3



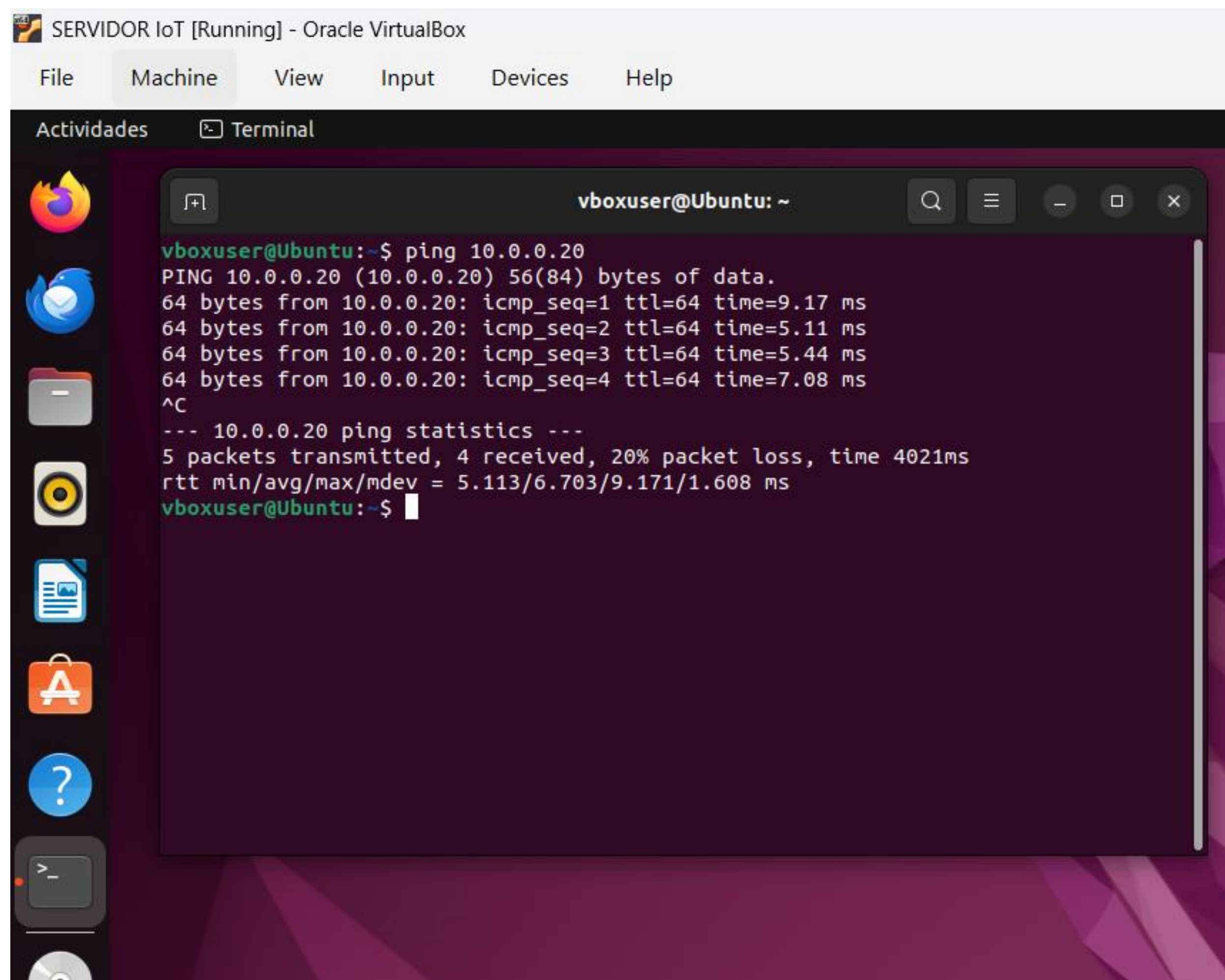
Nota. Prueba de ping entre Switches. *Fuente*. Autoria propia.

Todos se encuentran en la misma red, todo es ARP más switching.

La siguiente prueba de Servidor IoT a Controlador SDN.

Figura 16

Ping de Servidor IoT a Controlador SDN



Nota. Prueba de ping entre servidores. *Fuente*. Autoria propia.

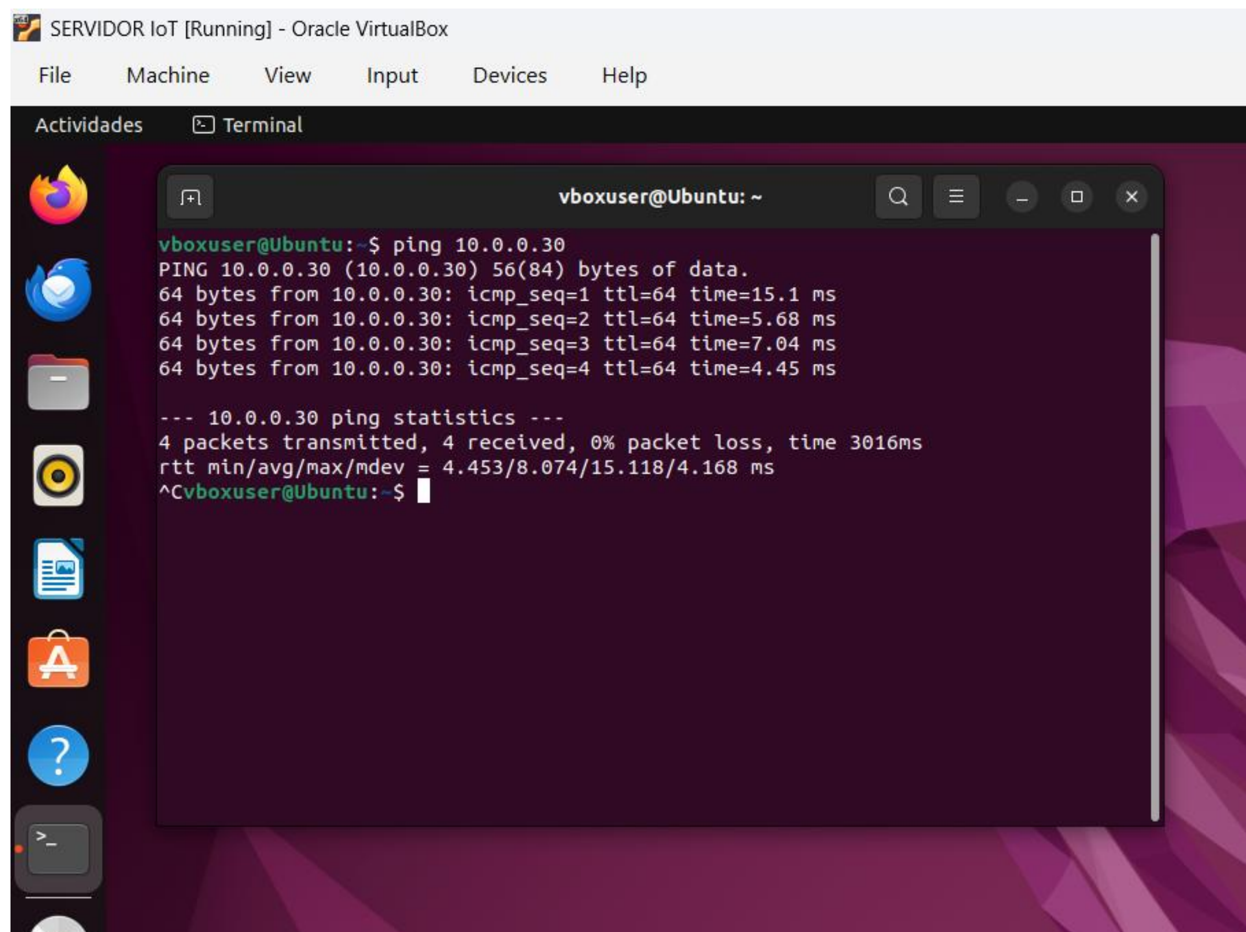
Aquí el Servidor IoT ve que está en 10.0.0.0/24, hace ARP, SW3 aprende la MAC, OVS reenvía.

Es importante tener en cuenta en este casp que el tráfico cruza varios switches, pero sigue siendo capa 2.

Por último, del Servidor IoT a Servidor de trazabilidad.

Figura 17

Ping de Servidor IoT a Servidor de trazabilidad



Nota. Prueba de ping entre servidores. *Fuente*. Autoria propia.

Es el mismo principio, no hay salto de red, no hay router solo switching distribuido.

La conectividad observada no depende de ningún protocolo de enrutamiento, ya que todos los dispositivos pertenecen a una misma subred IP. En este contexto, los hosts consideran cualquier dirección dentro del rango como directamente alcanzable, enviando tramas mediante resolución ARP. El tráfico es transportado a través de los switches Open vSwitch, los cuales operan en modo capa 2 (acción NORMAL), aprendiendo direcciones MAC y reenviando tramas sin necesidad de decisiones de enrutamiento.

Ahora se procederá con la configuración del controlador SDN, en SW1, SW2 y SW3.

Figura 18

Configuración OpenFlow

Nota. Configuración de OpenFlow en SW1, SW2 y Sw3. *Fuente*. Autoria propia.

Aquí los switches fueron configurados para establecer comunicación con el controlador SDN a través del protocolo OpenFlow 1.3, permitiendo la gestión centralizada de las decisiones de reenvío, se puede ver que los tres Switches tienen el “Controller tcp:10.0.0.20:6633”.

Ahora en el controlador SDN se empleará el controlador Ryu, una plataforma basada en Python que permite la implementación de aplicaciones SDN mediante OpenFlow.

Figura 19

Entorno OpenFlow en controlador SDN

CONTROLADOR SDN [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Actividades Terminal

vboxuser@Ubuntu: ~

```
vboxuser@Ubuntu:~$ source ryu-env/bin/activate
(ryu-env) vboxuser@Ubuntu:~$ which ryu-manager
/home/vboxuser/ryu-env/bin/ryu-manager
(ryu-env) vboxuser@Ubuntu:~$ ryu-manager --verbose ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7548e0d43ac0> ad
dress:('10.0.0.1', 48790)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7548e0d35a60>
```

Nota. OpenFlow en controlador SDN. *Fuente*. Autoria propia.

Se inicia el controlador SDN basado en Ryu mediante la aplicación simple_switch_13, la cual permite implementar un comportamiento de conmutador de aprendizaje (learning switch) utilizando OpenFlow 1.3. En la consola se observa la carga de los módulos necesarios y la inicialización del controlador, lo cual indica que el plano de control está activo y listo para gestionar los dispositivos de red.

Cada connected socket representa un switch OpenFlow conectado.

datapath_id identifica de forma única cada switch.

El controlador entra en.

Config mode a negociación inicial.

Main mode a operación normal.

Esto corresponde al establecimiento del canal de control en SDN (Plano de control con Plano de datos).

Figura 20

Conexiones en OpenFlow


```
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7548e0d43ac0> ad
dress:('10.0.0.1', 48790)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7548e0d35a60>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x18fe924e,OFPSwitc
hFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796748748670,n_buffers=0,n
_tables=254)
move onto main mode
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7548e27d5e50> ad
dress:('10.0.0.3', 43170)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7548e0d601f0>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xd427c7d8,OFPSwitc
hFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796747776807,n_buffers=0,n
_tables=254)
move onto main mode
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7548e0d3ac10> ad
dress:('10.0.0.2', 57044)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7548e0d60a00>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x4e145489,OFPSwitc
hFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796758785135,n_buffers=0,n
_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
```

Nota. Muestra de conexiones OpenFlow. *Fuente*. Autoria propia.

Se pueden observar las conexiones (connected socket) de los tres Switches 10.0.0.1, 10.0.0.2 y 10.0.0.3.

Una vez configurado el protocolo OpenFlow 1.3 en los switches de la topología (SW1, SW2 y SW3), se estableció la conexión con el controlador SDN implementado mediante Ryu, utilizando la dirección tcp:10.0.0.20:6633. Al iniciar el controlador con la aplicación simple_switch_13, se evidenció el establecimiento exitoso de la comunicación entre el plano de control y el plano de datos.

Durante la fase inicial, cada switch envía un mensaje de tipo OFPSwitchFeatures al controlador, el cual permite identificar sus capacidades, número de tablas de flujo y su identificador único (datapath ID). Este proceso confirma la correcta integración de los dispositivos dentro de la arquitectura SDN.

Posteriormente, al generar tráfico mediante un ping entre el servidor IoT (10.0.0.10) y el servidor de trazabilidad (10.0.0.30), se observaron múltiples eventos EventOFPPacketIn en la consola del controlador. Estos eventos indican que los switches no poseen reglas de flujo predefinidas para manejar los paquetes entrantes, por lo que los envían al controlador para su procesamiento.

Inicialmente, se identifican tramas con dirección MAC de destino ff:ff:ff:ff:ff:ff, correspondientes a mensajes ARP de difusión, utilizados para la resolución de direcciones IP a direcciones MAC. Este comportamiento es esperado, ya que los dispositivos requieren conocer la dirección física del destino antes de iniciar la comunicación.

Una vez resuelta la dirección MAC, se observan paquetes unicast entre las direcciones origen y destino, correspondientes al tráfico ICMP del ping. En este punto, el controlador Ryu, mediante la aplicación simple_switch_13, aprende dinámicamente las direcciones MAC y

procede a instalar reglas de flujo en los switches, permitiendo el reenvío directo de los paquetes sin necesidad de intervención adicional del controlador.

Figura 21

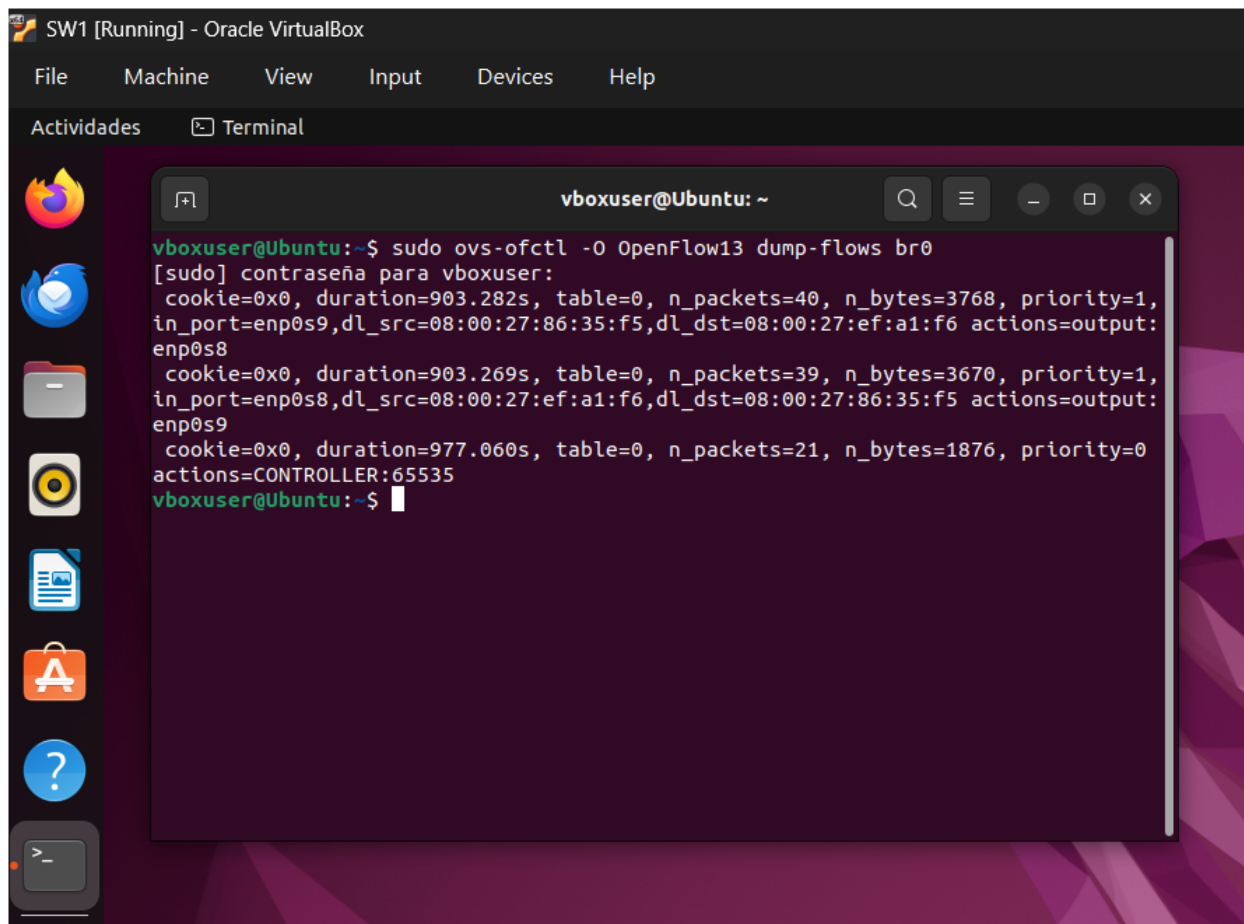
Eventos EventOFPPacketIn

Nota. Eventos generados en OpenFlow. *Fuente*. Autoria propia.

Ahora se realiza la verificación de los flujos instalados (Flows), en uno de los Switches, este caso SW1.

Figura 22

Flows instalados SW1



Nota. Flows de SW1. *Fuente*. Autoria propia.

Con el fin de verificar la instalación de reglas de flujo en los switches, se utilizó el comando ovs-ofctl dump-flows. Los resultados muestran las entradas en la tabla de flujo del Switch, incluyendo los criterios de coincidencia (match), contadores de paquetes y las acciones asociadas (actions).

Estas reglas son instaladas dinámicamente por el controlador Ryu en respuesta a los eventos PacketIn, permitiendo que los paquetes posteriores sean reenviados directamente por los switches sin necesidad de intervención del controlador.

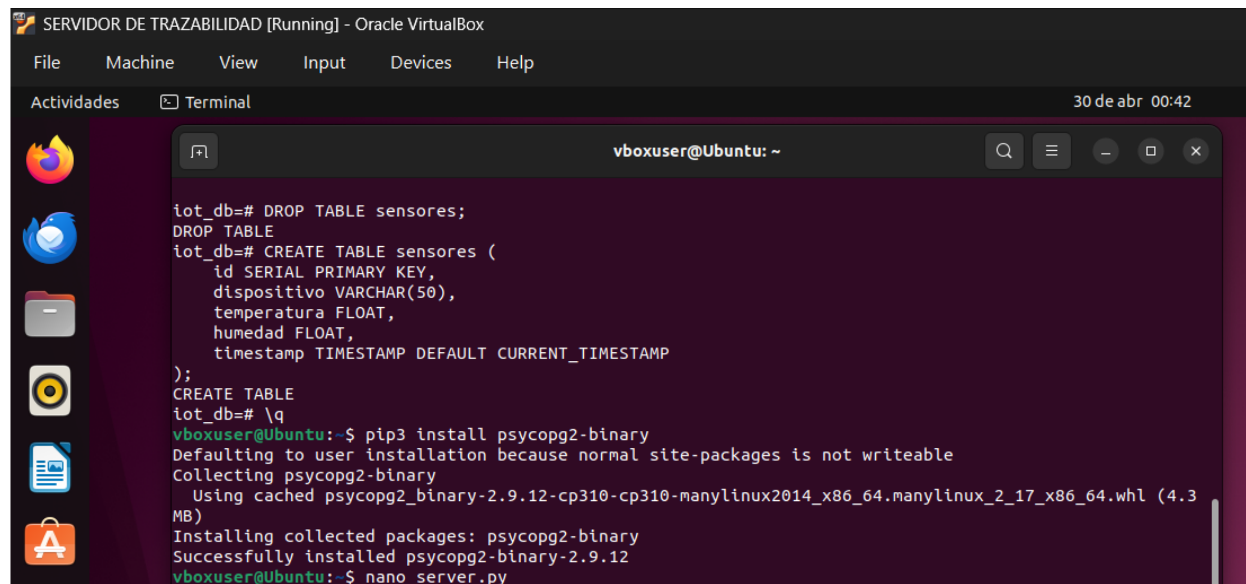
Este comportamiento reduce la carga del plano de control y optimiza el rendimiento de la red, evidenciando el funcionamiento del modelo SDN basado en OpenFlow.

Se desarrolló un servidor de trazabilidad basado en Flask y PostgreSQL, encargado de recibir, almacenar y gestionar los datos provenientes del servidor IoT. Este servidor expone una API REST que permite la recepción de datos en formato JSON y su almacenamiento en una base de datos estructurada, facilitando su posterior análisis.

Se creó una tabla con PostgreSQL donde se guardan los datos simulados IoT con un Timestamp automático.

Figura 23

Tabla PostgreSQL



Nota. Muestra de tabla PostgreSQL. *Fuente*. Autoria propia.

Luego se creó una API con Flask.

Figura 24

API de Servidor de trazabilidad

```
SERVIDOR DE TRAZABILIDAD [Running] - Oracle VirtualBox
File  Machine  View  Input  Devices  Help
Actividades  Terminal                                 30 de abr 00:39
vboxuser@Ubuntu: ~
  GNU nano 6.2                      server.py *
from flask import Flask, request, jsonify
import psycopg2

app = Flask(__name__)

def conectar():
    return psycopg2.connect(
        dbname="iot_db",
        user="postgres",
        password="postgres",
        host="localhost"
    )

@app.route('/data', methods=['POST'])
def recibir_datos():
    data = request.json

    dispositivo = data.get("dispositivo")
    temperatura = data.get("temperatura")
    humedad = data.get("humedad")

    conn = conectar()
    cur = conn.cursor()

    cur.execute(
        "INSERT INTO sensores (dispositivo, temperatura, humedad) VALUES (%s, %s, %s)",
        (dispositivo, temperatura, humedad)
    )

    conn.commit()
    cur.close()
    conn.close()

    return jsonify({"status": "ok"}), 200

@app.route('/data', methods=['GET'])
def ver_datos():
    conn = conectar()
    cur = conn.cursor()

^G Ayuda    ^O Guardar     ^W Buscar     ^K Cortar  ^T Ejecutar    ^C Ubicación   M-U Deshacer
^X Salir    ^R Leer fich.  ^\ Reemplazar ^U Pegar   ^J Justificar  ^/ Ir a línea  M-E Rehacer
```

Nota. Código desarrollado de la API. *Fuente*. Autoria propia.

Aquí.

/data POST va a recibir datos IoT.

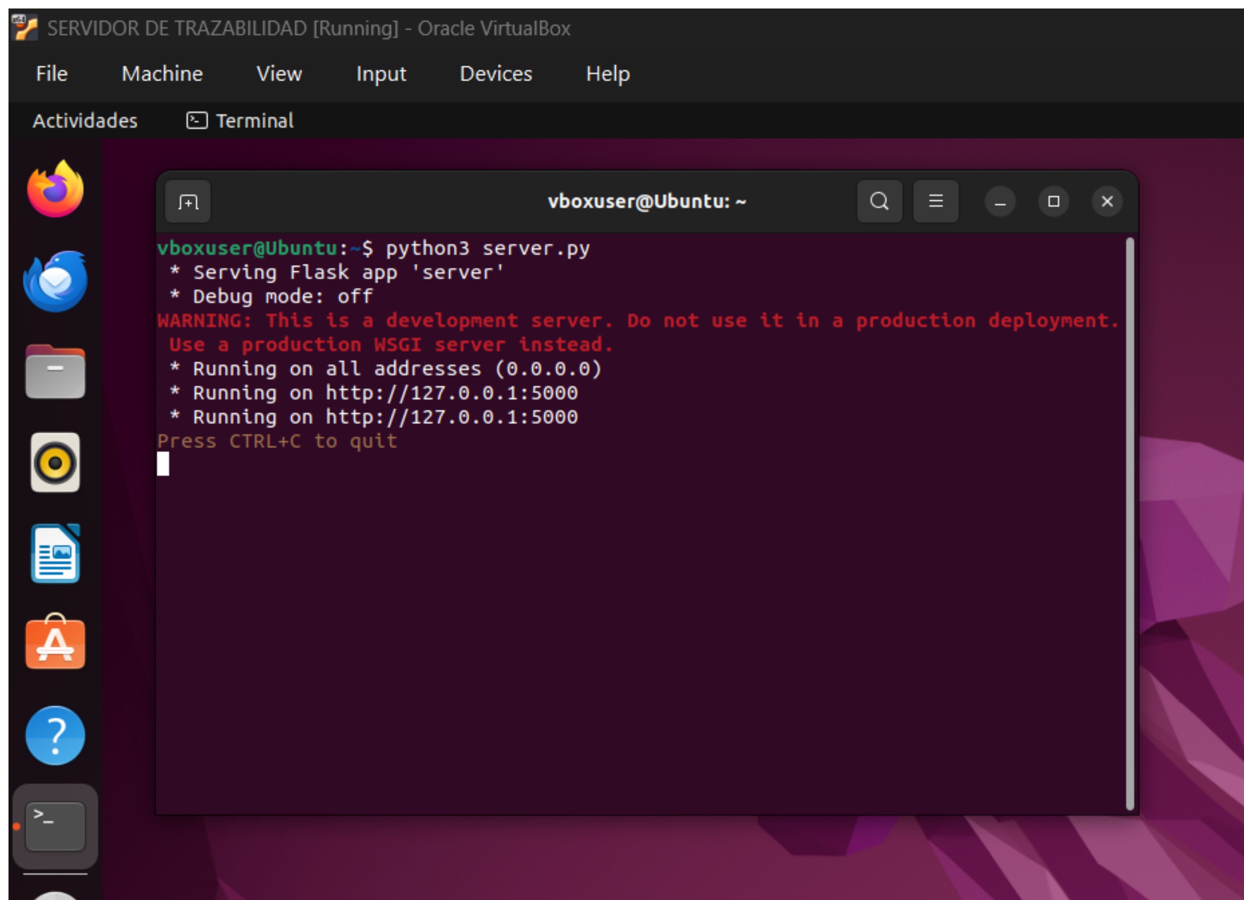
/data GET va a consultar datos.

Inserta en PostgreSQL.

Se ejecuta el archivo de Python iot_sender.py.

Figura 25

Ejecución de código para recibir datos



Nota. Servidor de trazabilidad listo para recibir datos. *Fuente*. Autoria propia.

Se puede ver que el Flask está ejecutándose correctamente.

El servidor está.

Escuchando en el puerto 5000.

Disponible en todas las interfaces de red (0.0.0.0).

Aunque aparece.

http://127.0.0.1:5000.

El servidor también está disponible en.

http://10.0.0.30:5000.

El servidor de trazabilidad fue implementado utilizando el framework Flask en Python, el cual permite la creación de servicios web ligeros para la recepción de datos en formato JSON.

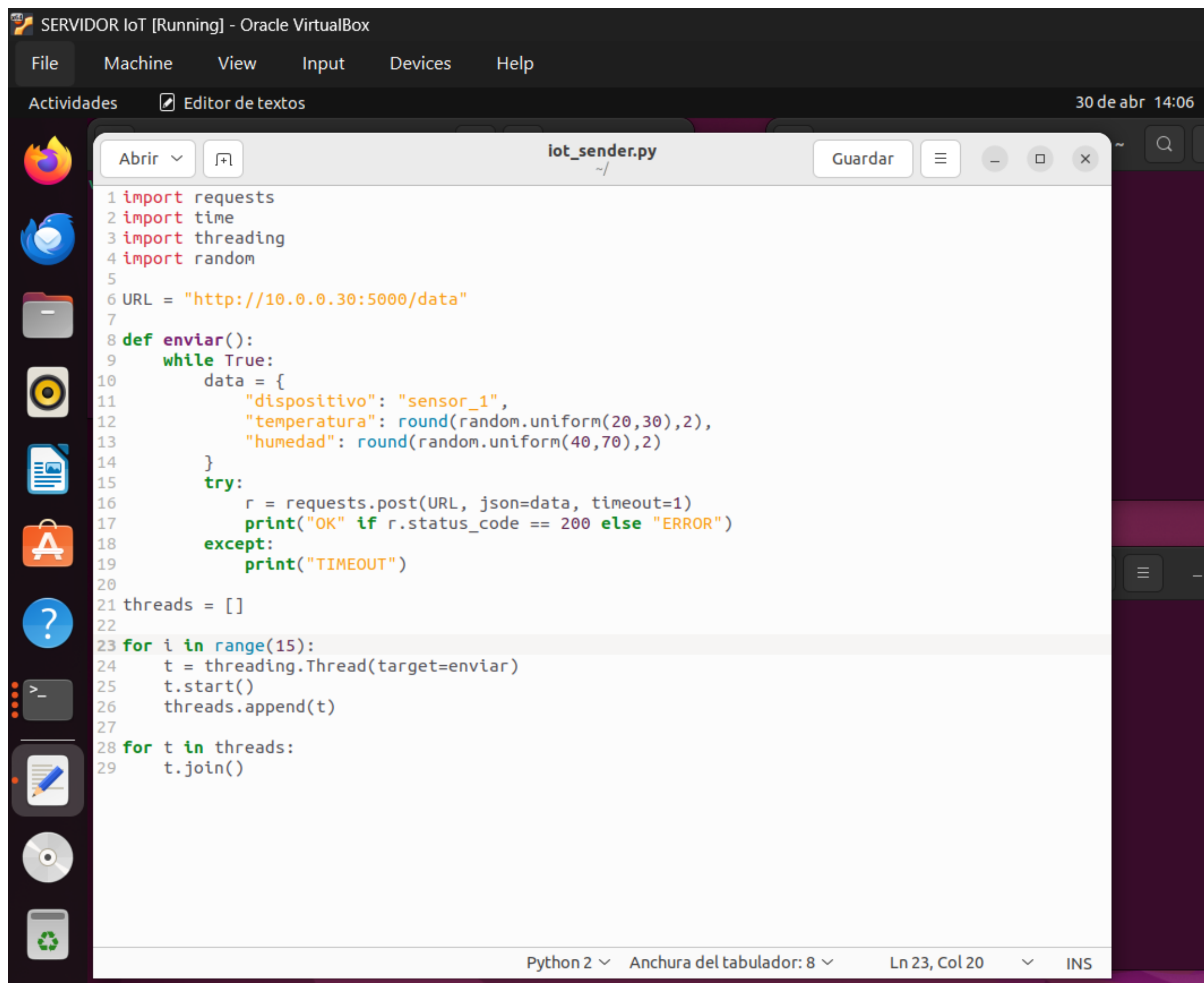
El servidor se configuró para escuchar en el puerto 5000 y en todas las interfaces de red (0.0.0.0), lo que permite la recepción de datos provenientes de otros nodos de la red, específicamente del servidor IoT.

Al ejecutar el servidor, se observa su correcto funcionamiento mediante la inicialización del servicio web y la disponibilidad del endpoint /data, el cual permite tanto la recepción (método POST) como la consulta (método GET) de los datos generados por los dispositivos IoT simulados.

Posteriormente, se implementó el servidor IoT por medio de un generador de datos IoT utilizando Python, el cual simula el comportamiento de sensores mediante la generación periódica de variables como temperatura y humedad. Estos datos son enviados al servidor de trazabilidad a través de peticiones HTTP, permitiendo emular un flujo continuo de información en la red.

Figura 26

Código Python para envío de datos



Nota. Muestra de código de Python desarrollado en servidor IoT. *Fuente*. Autoria propia.

Se realiza la prueba del envío de datos, por medio de la dirección IP del servidor de trazabilidad y por el puerto 5000, se puede ver los valores aleatorios de temperatura, y la respuesta.

Figura 27

Envío de datos desde el servidor IoT

```
SERVIDOR IoT [Running] - Oracle VirtualBox
File   Machine   View   Input   Devices   Help
Actividades   Terminal                                   30 de abr 11:08
vboxuser@Ubuntu: ~
vboxuser@Ubuntu:~$ curl http://10.0.0.30:5000/data
[]
vboxuser@Ubuntu:~$ python3 iot_sender.py
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 27.27, 'humedad': 46.08} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 20.89, 'humedad': 43.74} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 28.84, 'humedad': 64.31} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 27.17, 'humedad': 66.07} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 26.04, 'humedad': 48.68} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 23.81, 'humedad': 65.39} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 24.81, 'humedad': 66.2} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 24.61, 'humedad': 41.57} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 28.83, 'humedad': 42.35} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 24.96, 'humedad': 60.73} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 28.93, 'humedad': 58.74} Respuesta: 200
Enviado: {'dispositivo': 'sensor_1', 'temperatura': 24.07, 'humedad': 52.29} Respuesta: 200
^CTraceback (most recent call last):
  File "/home/vboxuser/iot_sender.py", line 20, in <module>
    time.sleep(2)
KeyboardInterrupt

vboxuser@Ubuntu:~$
```

Nota. Prueba de envío de datos. *Fuente*. Autoria propia.

Como parte de la arquitectura propuesta, se implementó un dashboard web sencillo integrado en el servidor de trazabilidad, con el objetivo de visualizar en tiempo real los datos generados por los sensores IoT. Esta funcionalidad se desarrolló utilizando el framework Flask, aprovechando la misma API encargada de recibir y almacenar la información en la base de datos PostgreSQL.

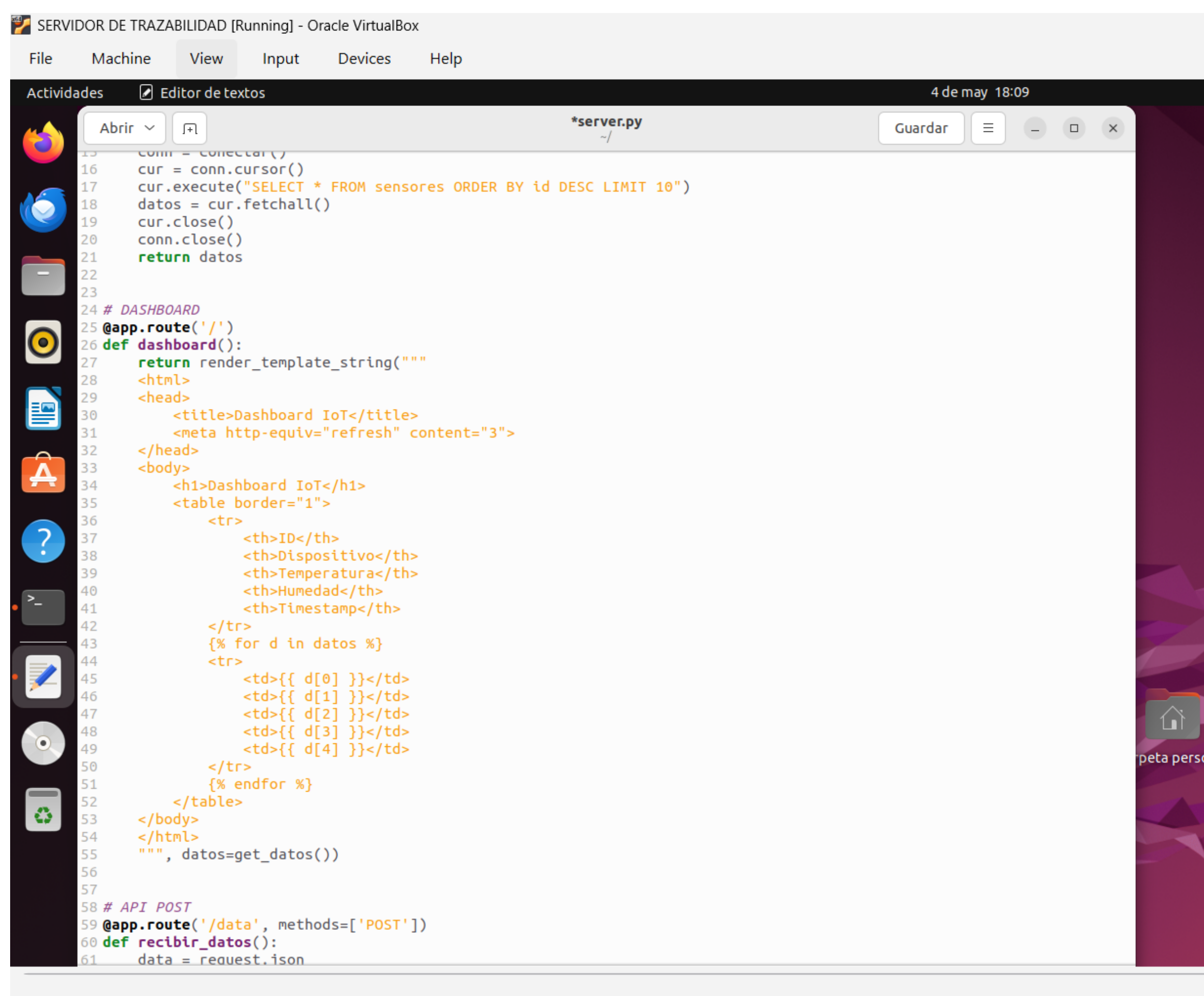
A nivel de implementación, se añadió una nueva ruta (/) en el servidor, la cual consulta los registros más recientes almacenados en la base de datos mediante una función auxiliar. Estos datos son posteriormente renderizados en una plantilla HTML generada dinámicamente, utilizando render_template_string, lo que permite construir una interfaz sin necesidad de

archivos adicionales. La información se organiza en una tabla que incluye identificador, dispositivo, temperatura, humedad y marca de tiempo.

Adicionalmente, se configuró una actualización automática del dashboard cada pocos segundos mediante una etiqueta HTML (meta refresh), lo que permite reflejar en tiempo casi real los datos que están siendo enviados desde el servidor IoT, sin requerir intervención del usuario.

Figura 28

Código de Dashboard



Nota. Muestra de código desarrollado para el Dashboard. *Fuente*. Autoria propia.

Se puede observar que los datos enviados desde el servidor IoT son recibidos correctamente por el servidor de trazabilidad y almacenados en la base de datos, reflejándose de manera inmediata en la interfaz web. La información se presenta en forma tabular, permitiendo visualizar de manera clara y ordenada los valores de temperatura y humedad generados por los sensores, junto con su respectivo identificador y timestamp.

La actualización automática del dashboard permite evidenciar el flujo continuo de datos, mostrando cómo los registros se van incorporando dinámicamente a medida que la aplicación IoT envía nuevas mediciones. Esto facilita la validación visual del sistema, confirmando no solo la correcta transmisión de datos a través de la red, sino también su procesamiento y almacenamiento en el backend.

Figura 29

Dashboard funcionando por puerto 5000

Nota. Prueba de funcionamiento de envío de datos con Dashboard. *Fuente*. Autoria propia.

A continuación, se iniciará con las pruebas experimentales.

Escenario 1 Red sin SDN

Cómo se ha explicado, en este momento se está usando OVS en los Switches y Ryu (simple_switch_13), esto es SDN, pero se encuentra en modo learning switch básico, no hay QoS, ni control, ni priorización.

Por lo que, en este escenario inicial, aunque la red opera bajo un controlador SDN, no se aplican políticas de gestión, por lo que el comportamiento equivale a un reenvío tradicional sin control de tráfico.

Figura 30

Prueba inicial de Ping y envío de datos desde Servidor IoT

Nota. Prueba de Ping y envío de datos en servidor IoT. *Fuente*. Autoria propia.

Para la realización de las pruebas experimentales, fue necesario modificar el código original del generador de tráfico IoT con el fin de incorporar mecanismos de medición que permitieran cuantificar el desempeño de la comunicación bajo condiciones de congestión de red. En este sentido, se añadieron contadores globales (ok y fail) que registran respectivamente el número de solicitudes HTTP exitosas (código de respuesta 200) y aquellas que no logran completarse, ya sea por errores de red o por expiración del tiempo de espera (timeout).

Asimismo, se integró un mecanismo de sincronización mediante el uso de un Lock, garantizando la correcta actualización de estas variables en un entorno concurrente con múltiples hilos. Adicionalmente, se implementó un hilo de monitoreo que calcula periódicamente la tasa de éxito de las comunicaciones, definida como la proporción de solicitudes exitosas respecto al total de solicitudes enviadas. Esta métrica permitió obtener un valor promedio representativo del comportamiento del sistema, el cual fue utilizado para comparar el desempeño de la red en escenarios sin implementación de SDN.

El código implementa un generador de tráfico IoT basado en múltiples hilos de ejecución, cuyo objetivo es simular el comportamiento concurrente de varios dispositivos enviando datos hacia un servidor HTTP. Para ello, se emplea la librería threading, que permite crear múltiples instancias de la función enviar(), cada una representando un dispositivo IoT independiente. Estas instancias generan datos aleatorios (temperatura y humedad) y los envían mediante solicitudes HTTP POST al servidor definido en la variable URL. Se incorpora además un mecanismo de control de concurrencia mediante un Lock, con el fin de evitar condiciones de carrera al momento de actualizar variables compartidas (ok y fail), las cuales contabilizan las solicitudes exitosas y fallidas respectivamente.

Adicionalmente, se introduce una función de monitoreo (monitor()), ejecutada en un hilo independiente, que cada cinco segundos calcula métricas clave del sistema, como el total de solicitudes enviadas y la tasa de éxito en porcentaje. Esta funcionalidad permite observar en tiempo real el comportamiento del sistema bajo condiciones de carga y congestión de red. La inclusión de un tiempo de espera (timeout=1) en las solicitudes HTTP es fundamental, ya que permite detectar fallos de comunicación (timeouts), los cuales son contabilizados como errores,

proporcionando así una métrica más precisa del desempeño del sistema frente a condiciones adversas de red.

Figura 31

Código modificado para envío de datos por HTTP con porcentaje de error

Nota. Modificación de código con porcentaje de error. *Fuente*. Autoria propia.

Se realizó otra prueba enviando nuevamente un ping y también datos por HTTP para ver el porcentaje de error y el comportamiento de la red.

Figura 32

Prueba inicial sin carga de red

Nota. Prueba de Ping sin carga. *Fuente*. Autoria propia.

Como se pudo observar en la prueba inicial, la red no presenta condiciones de estrés significativas, debido a que únicamente se ejecutan dos tipos de tráfico: solicitudes ICMP mediante el comando ping y el envío de datos a través del script IoT (iot_sender.py). En este escenario, no existen flujos concurrentes de alto volumen ni congestión en los enlaces, lo que permite que los paquetes se transmitan sin retrasos relevantes ni pérdidas.

Desde el punto de vista técnico, los resultados evidencian una latencia promedio baja (alrededor de 8 ms), cero pérdida de paquetes (0%) y una tasa de éxito del 100% en las solicitudes HTTP hacia el servidor de trazabilidad. Esto indica que la red se encuentra en condiciones óptimas de operación, con suficiente capacidad disponible, sin colas de congestión ni descartes de paquetes. En consecuencia, este escenario sirve como línea base para comparar posteriormente el comportamiento de la red bajo condiciones de carga y estrés.

En la máquina correspondiente al servidor de trazabilidad se abrieron tres terminales para simular distintos servicios de red operando simultáneamente:

Servidor HTTP (API Flask)

python3 server.py

Este comando inicia una aplicación web basada en Flask, la cual actúa como punto de recepción de datos enviados por los dispositivos IoT. El servidor escucha en la dirección http://10.0.0.30:5000/data y responde a las solicitudes HTTP tipo POST. Cada respuesta exitosa genera un código de estado 200, lo que permite medir la tasa de éxito del sistema bajo diferentes condiciones de red.

Servidor Iperf3 TCP

iperf3 -s

Este comando levanta un servidor de pruebas de rendimiento utilizando el protocolo TCP en el puerto por defecto (5201). Su función es recibir conexiones desde clientes iperf y medir el throughput de la red, permitiendo evaluar el comportamiento del control de congestión y la eficiencia del transporte bajo carga.

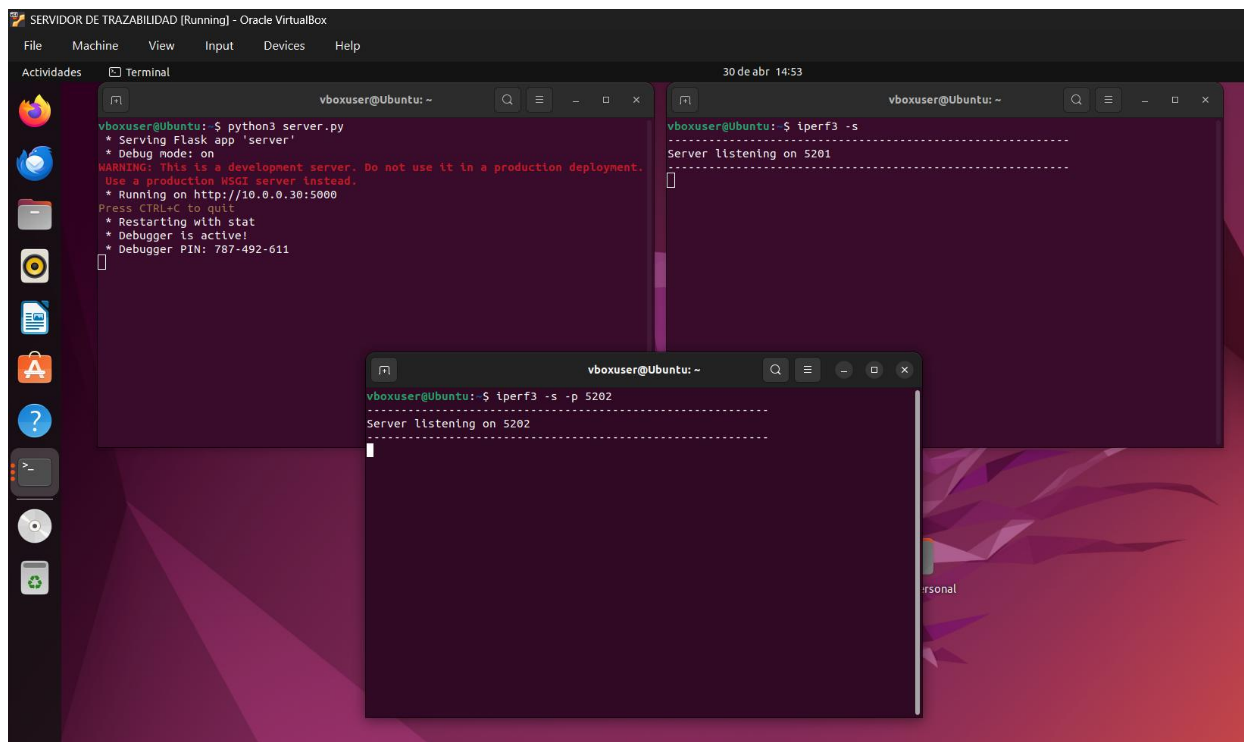
Servidor Iperf3 UDP

iperf3 -s -p 5202

En esta terminal se inicia un servidor iperf3 adicional, pero utilizando el protocolo UDP en el puerto 5202. A diferencia de TCP, UDP no implementa control de congestión ni retransmisión, por lo que permite analizar la pérdida de paquetes, jitter y comportamiento de la red bajo tráfico no confiable y de alta velocidad.

Figura 33

Habilitación de servidores



Nota. Servidores escuchando. *Fuente*. Autoria propia.

Generación de Tráfico Servidor IoT

En la máquina del servidor IoT se abrieron cinco terminales, ejecutando simultáneamente diferentes tipos de tráfico con el fin de generar carga en la red.

Simulación de Dispositivos IoT

python3 iot_sender.py

Este script simula múltiples dispositivos IoT enviando datos continuamente al servidor mediante solicitudes HTTP POST. Además, registra estadísticas de éxito y fallo, lo que permite cuantificar el impacto de la congestión en aplicaciones reales.

Tráfico TCP Intensivo

iperf3 -c 10.0.0.30 -t 120 -P 10

Este comando genera tráfico TCP hacia el servidor durante 120 segundos utilizando 10 flujos paralelos (-P 10). Esto incrementa significativamente la carga en la red y permite evaluar el comportamiento del throughput, así como la aparición de retransmisiones debido a congestión.

Tráfico UDP de Alto Volumen

iperf3 -c 10.0.0.30 -u -b 200M -p 5202 -t 120

Se genera tráfico UDP a una tasa constante de 200 Mbps durante 120 segundos hacia el servidor, esto provoca saturación del enlace y permite medir pérdida de paquetes y degradación del rendimiento bajo de alta carga.

Medición de Latencia

ping 10.0.0.30

Este comando envía paquetes ICMP periódicos para medir la latencia de la red. Permite observar cómo el tiempo de respuesta aumenta cuando la red entra en estado de congestión.

Ping con Carga (Flood)

sudo ping 10.0.0.30 -f -s 1400

Este comando genera tráfico ICMP de alta intensidad (modo flood) con paquetes grandes (1400 bytes), aumentando aún más la carga en la red. Su propósito es estresar el enlace y contribuir a la congestión, afectando directamente la latencia y la pérdida de paquetes.

En el escenario sin SDN se pudo observar un comportamiento altamente ineficiente de la red bajo condiciones de carga simultánea. La ejecución concurrente de tráfico IoT, pruebas de rendimiento TCP y UDP, así como pruebas de latencia mediante ping, generó una saturación significativa en los enlaces. Esto se tradujo en una alta pérdida de paquetes, incremento considerable de la latencia y una tasa de éxito muy baja en la aplicación IoT.

Ventana 1 Python3 iot_sender.py

La aplicación IoT presentó una tasa de éxito extremadamente baja, con valores que oscilaron aproximadamente entre 2% y 7%, lo que implica que más del 90% de las solicitudes HTTP fallaron. Esto indica que la red no fue capaz de garantizar la entrega de datos críticos, debido principalmente a la congestión generada por el tráfico concurrente. Los timeouts definidos en el código (timeout=1) contribuyeron a evidenciar la incapacidad de la red para responder oportunamente.

Ventana 2 Iperf3 -c 10.0.0.30 -t 120 -P 10 (TCP)

El tráfico TCP presentó inestabilidad significativa, evidenciada por errores como "control socket has closed unexpectedly", lo que indica que las conexiones no pudieron mantenerse activas. El throughput efectivo fue inconsistente y bajo, debido a retransmisiones constantes provocadas por pérdida de paquetes. TCP intentó adaptarse mediante control de congestión, pero la saturación de la red impidió un rendimiento estable.

Ventana 3 Iperf3 -c 10.0.0.30 -u -b 200M -p 5202 -t 120 (UDP)

Desde el lado cliente, el tráfico UDP reportó una tasa de envío constante cercana a 200 Mbps, con pérdida aparentemente nula. Sin embargo, esto es característico del protocolo UDP, que no implementa control de congestión. En realidad, este comportamiento contribuyó a saturar la red, generando pérdida masiva en el destino, lo que afecta directamente al resto de aplicaciones.

Ventana 4 Ping 10.0.0.30

Las pruebas de ping mostraron una latencia promedio cercana a 266 ms, con picos superiores a 300 ms y una pérdida de paquetes aproximada del 60%–65%. Estos valores son extremadamente altos para una red local, donde normalmente se esperan tiempos de respuesta

del orden de pocos milisegundos y pérdidas de paquetes prácticamente nulas, indicando congestión severa y colas saturadas en los dispositivos intermedios.

Ventana 5 Ping -f -s 1400 (Flood Ping)

La prueba de flood ping intensificó la saturación de la red, generando pérdidas de paquetes cercanas al 60% y latencias elevadas. Este comportamiento simula condiciones de ataque o sobrecarga, donde la red colapsa completamente al no contar con mecanismos de control.

Figura 34

Resultados sin SDN en servidor IoT

Nota. Ventanas de resultados sin SND. *Fuente*. Autoria propia.

Ventana 1 Python3 server.py

El servidor Flask se mantuvo operativo y respondió correctamente con códigos HTTP 200. Sin embargo, la cantidad de solicitudes recibidas fue significativamente menor a las enviadas, lo que confirma que la pérdida de paquetes ocurrió en la red y no en la aplicación.

Ventana 2 Iperf3 -s (TCP)

El servidor reportó múltiples interrupciones en las conexiones TCP, con sesiones incompletas y bajo throughput efectivo. Esto confirma que el canal de comunicación estaba altamente degradado, impidiendo la transmisión estable de datos.

Ventana 3 Iperf3 -s -p 5202 (UDP)

El servidor UDP evidenció la realidad del tráfico: aunque el cliente enviaba 200 Mbps, el servidor solo recibía aproximadamente 30–40 Mbps, con pérdidas cercanas al 80%. Esto confirma una saturación extrema del canal.

Figura 35

Resultados sin SDN en servidor de trazabilidad

Nota. Ventanas de resultados sin SDN en servidor de trazabilidad. *Fuente*. Autoria propia.

Controlador SDN

El controlador operó como un switch L2 básico, basado en aprendizaje MAC y flooding inicial. Se observó un alto volumen de eventos Packet-In, lo que indica que muchos paquetes debieron ser procesados por el controlador antes de instalar flujos, aquí se pudo ver que:

No hay priorización.

No hay control de tráfico.

El controlador no mitiga congestión.

La red se comporta como una red tradicional.

Figura 36 *Resultados sin SDN en controlador SDN*

Nota. Resultados sin SDN en controlador. *Fuente*. Autoria propia.

Escenario 2 Red con SDN

En este escenario se implementa una red basada en el paradigma de Redes Definidas por Software (SDN), en la cual el plano de control se desacopla del plano de datos y es centralizado en un controlador. A diferencia del escenario tradicional, donde cada switch toma decisiones de forma distribuida, en SDN las decisiones de encaminamiento y políticas de red son gestionadas de manera centralizada.

El objetivo principal de este escenario es evaluar el comportamiento de la red al aplicar políticas de calidad de servicio (QoS) definidas dinámicamente desde el controlador. En particular, se busca.

Priorizar el tráfico generado por dispositivos IoT.

Controlar el tráfico intensivo (como pruebas de carga con TCP).

Limitar el tráfico no crítico o potencialmente innecesario (UDP masivo).

Para ello, se implementa un controlador basado en Ryu que interactúa con switches Open vSwitch mediante el protocolo OpenFlow 1.3, permitiendo la instalación dinámica de reglas de flujo según el tipo de tráfico detectado.

Inicialización del Controlador

Antes de ejecutar la aplicación SDN, se ejecutan los siguientes comandos en el controlador SDN en ese orden.

Activa el entorno virtual de Python donde se encuentra instalado el framework Ryu. Esto garantiza que se utilicen las versiones correctas de librerías y dependencias.

Desactiva el uso de DNS asíncrono en la librería eventlet.

Esto evita conflictos conocidos entre eventlet y versiones recientes de Python, asegurando la estabilidad del controlador.

Ejecuta el controlador SDN con ryu-manager: inicia el framework Ryu, luego Qos_controller.py: aplicación personalizada y --verbose: permite visualizar en tiempo real los eventos OpenFlow.

Esto es clave porque permite observar conexión de switches, eventos PacketIn y aplicación de políticas QoS.

Figura 37

Inicialización del controlador SDN

```
CONTROLADOR SDN [Running] - Oracle VirtualBox
File   Machine   View   Input   Devices   Help
Actividades   Terminal
vboxuser@Ubuntu: ~
vboxuser@Ubuntu:~$ source ryu-env/bin/activate
export EVENTLET_NO_GREENDNS=yes
ryu-manager --verbose qos_controller.py
loading app qos_controller.py
loading app ryu.controller.ofp_handler
instantiating app qos_controller.py of QoSController
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK QoSController
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'QoSController': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'QoSController': {'config'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x79a09a20af10> ad
dress:('10.0.0.3', 53246)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x79a09a20a610>
move onto config mode
```

Nota. Inicio de controlador SND. *Fuente*. Autoria propia.

A continuación, se presenta el código que se desarrollo para gestionar el tráfico:

Definición del Controlador

class QoSController(app_manager.RyuApp) y OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION], se define una aplicación Ryu que utiliza OpenFlow 1.3. Esto permite trabajar con reglas avanzadas de flujo y control granular del tráfico.

Inicialización

Con self.mac_to_port = {} se crea una tabla MAC puerto, que permite al controlador aprender dinámicamente la topología (igual que un switch tradicional).

Regla por Defecto (Table-Miss)

Para match = parser.OFPMatch() y actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)], esta regla indica que si un paquete no coincide con ninguna regla, se envía al controlador, esto es fundamental porque permite inspeccionar el tráfico permite tomar decisiones dinámicas y es la base del funcionamiento SDN.

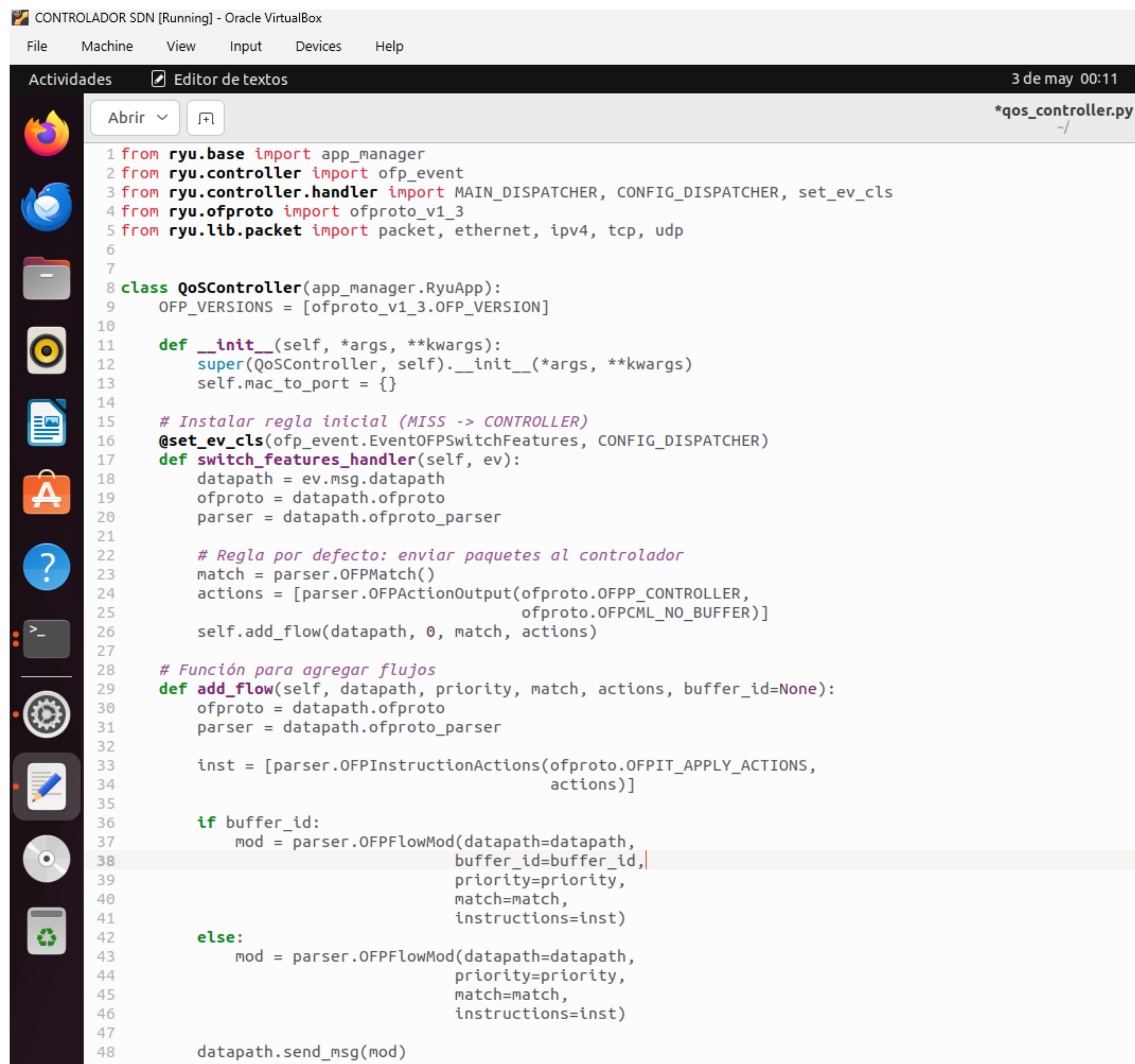
Función Add_flow

Encapsula la creación de reglas OpenFlow (FlowMod) y permite definir coincidencias (match), definir acciones (actions) y asignar prioridad.

Esto es clave porque las políticas QoS se implementan mediante reglas con diferentes prioridades.

Figura 38

Parte 1. Código qos_controller.py



Nota. Muestra de código de calidad del servicio desarrollado parte 1. *Fuente*. Autoria propia.

Procesamiento de Paquetes (PacketIn)

Se usa @set_ev_cls(EventOFPPacketIn, MAIN_DISPATCHER) cada vez que un switch no sabe qué hacer con un paquete y lo envía al controlador para que se ejecute esta función.

Aprendizaje de Direcciones MAC

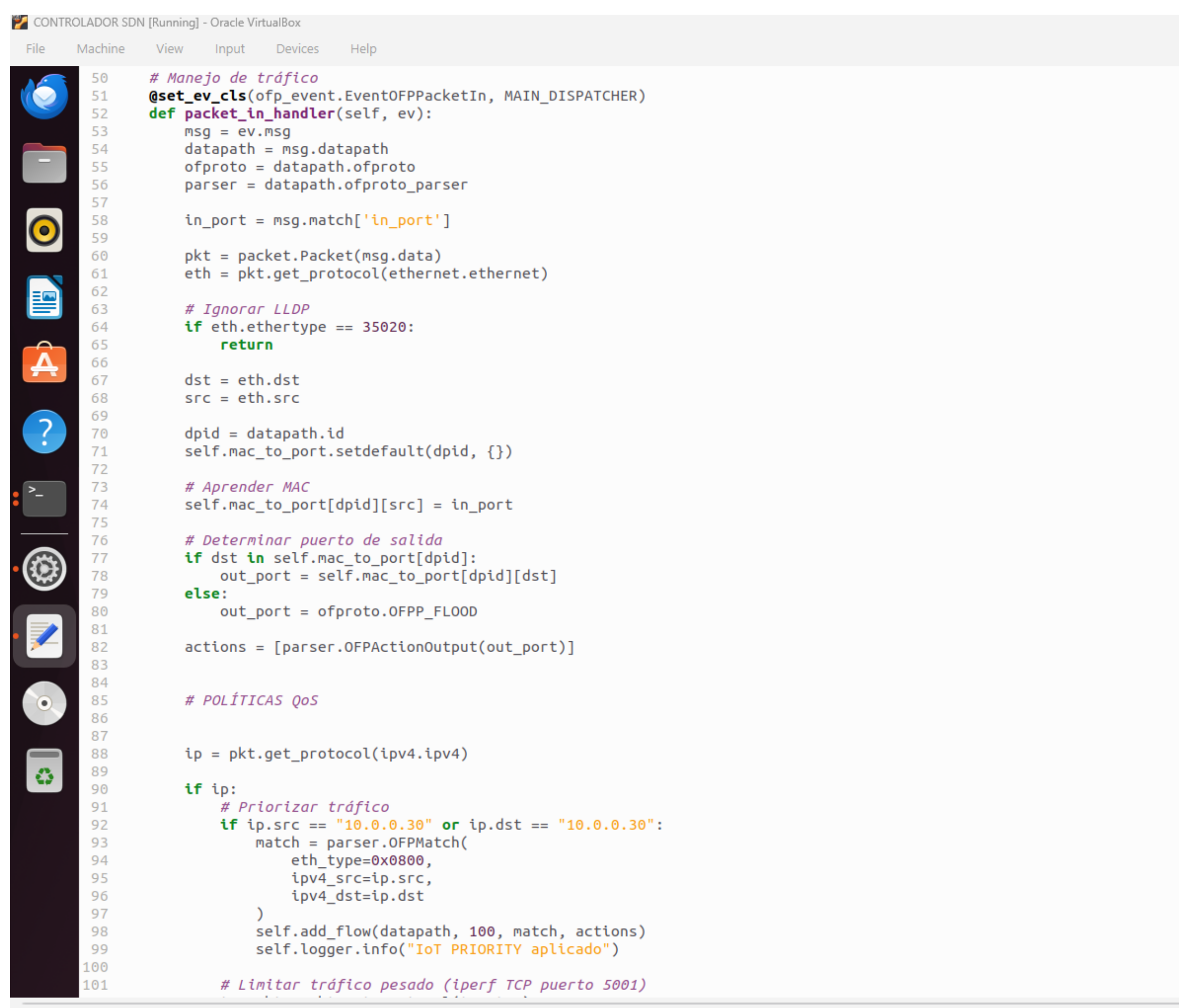
Con self.mac_to_port[dpid][src] = in_port, el controlador aprende, qué dispositivo está en qué puerto y permite evitar flooding innecesario.

Lógica de Reenvío (switch L2)

Con if dst in tabla se envía directo y luego else con FLOOD replica el comportamiento de un switch tradicional, pero controlado por software.

Figura 39

Parte 2. Código qos_controller.py



Nota. Muestra de código de calidad del servicio desarrollado parte 2. *Fuente*. Autoria propia.

Políticas de QoS y Priorización de Tráfico IoT

Con if ip.src == "10.0.0.30" or ip.dst == "10.0.0.30": y priority = 100, se asigna alta prioridad (100) y este tráfico se instala como regla en el switch

Inicialmente se puso esta IP para hacer una prueba inicial de ping entre SW1 a Controlador SND, pero para las pruebas del escenario 2 se asignará la IP del dispositivo IoT (10.0.0.10), con el fin de garantizar que el tráfico generado por sensores tenga mayor prioridad en la red, reduciendo latencia y pérdida de paquetes frente a tráfico convencional.

Control de Tráfico TCP

Con ip_proto=6 y priority=10 se detecta tráfico TCP (ej: iperf) y se le asigna prioridad media

No bloquea el tráfico, pero evita que domine la red.

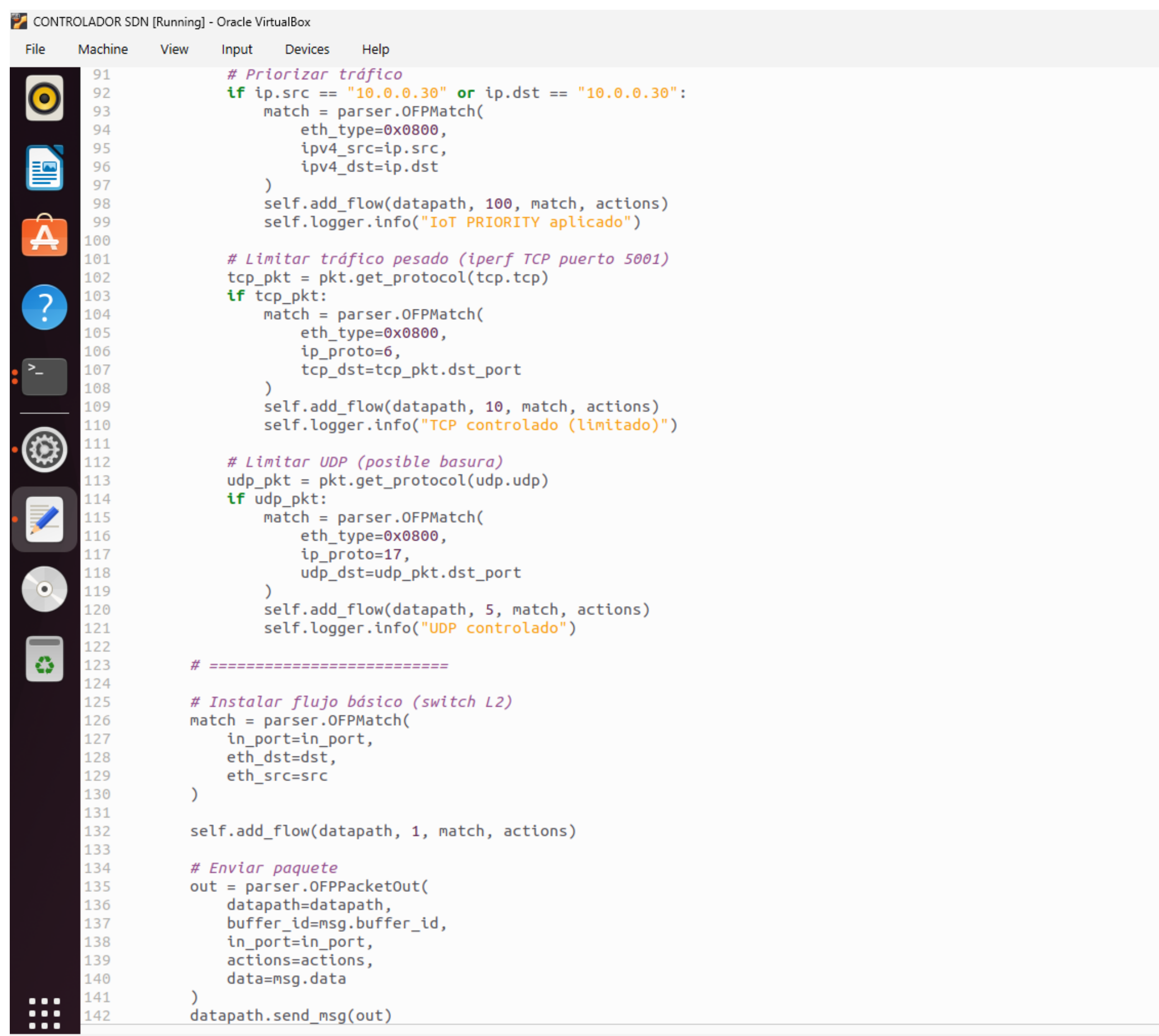
Control de Tráfico UDP

Con ip_proto=17 y priority=5 se considera tráfico potencialmente “basura” y se le asigna baja prioridad

Se atiende de último, reduciendo su impacto.

Figura 40

Parte 3. Código qos_controller.py




```
91          # Priorizar tráfico
92          if ip.src == "10.0.0.30" or ip.dst == "10.0.0.30":
93              match = parser.OFPMatch(
94                  eth_type=0x0800,
95                  ipv4_src=ip.src,
96                  ipv4_dst=ip.dst
97              )
98              self.add_flow(datapath, 100, match, actions)
99              self.logger.info("IoT PRIORITY aplicado")
100
101         # Limitar tráfico pesado (iperf TCP puerto 5001)
102         tcp_pkt = pkt.get_protocol(tcp.tcp)
103         if tcp_pkt:
104             match = parser.OFPMatch(
105                 eth_type=0x0800,
106                 ip_proto=6,
107                 tcp_dst=tcp_pkt.dst_port
108             )
109             self.add_flow(datapath, 10, match, actions)
110             self.logger.info("TCP controlado (limitado)")
111
112         # Limitar UDP (posible basura)
113         udp_pkt = pkt.get_protocol(udp.udp)
114         if udp_pkt:
115             match = parser.OFPMatch(
116                 eth_type=0x0800,
117                 ip_proto=17,
118                 udp_dst=udp_pkt.dst_port
119             )
120             self.add_flow(datapath, 5, match, actions)
121             self.logger.info("UDP controlado")
122
123     # ==========================
124
125     # Instalar flujo básico (switch L2)
126     match = parser.OFPMatch(
127         in_port=in_port,
128         eth_dst=dst,
129         eth_src=src
130     )
131
132     self.add_flow(datapath, 1, match, actions)
133
134     # Enviar paquete
135     out = parser.OFPPacketOut(
136         datapath=datapath,
137         buffer_id=msg.buffer_id,
138         in_port=in_port,
139         actions=actions,
140         data=msg.data
141     )
142     datapath.send_msg(out)
```

Nota. Muestra de código de calidad del servicio desarrollado parte 3. *Fuente*. Autoria propia.

En OpenFlow, las reglas con mayor prioridad se evalúan primero, como se puede observar en la siguiente tabla, así se definió en el código.

Tabla 3

Asignación de prioridades

Tipo de trafico	Prioridad
IoT	100
TCP	10
UDP	5
Default	1

Nota. Esta tabla muestra la asignación de las prioridades para el tráfico en la red con SDN.

Fuente. Autoria propia.

Luego de iniciar el controlador SDN se deben ejecutar los siguientes comandos en cada Switch con sudo ovs-ofctl -O OpenFlow13 del-flows br0 y con sudo ovs-ofctl -O OpenFlow13 add-flow br0 priority=0,actions=CONTROLLER:65535.

Figura 41

Eliminación de reglas y prueba de ping

```
SW1 [Running] - Oracle VirtualBox
File   Machine   View   Input   Devices   Help
Actividades   Terminal
vboxuser@Ubuntu: ~
vboxuser@Ubuntu:~$ sudo ovs-ofctl -O OpenFlow13 del-flows br0
sudo ovs-ofctl -O OpenFlow13 add-flow br0 priority=0,actions=CONTROLLER:65535
vboxuser@Ubuntu:~$ ping 10.0.0.30
PING 10.0.0.30 (10.0.0.30) 56(84) bytes of data.
64 bytes from 10.0.0.30: icmp_seq=1 ttl=64 time=47.2 ms
64 bytes from 10.0.0.30: icmp_seq=2 ttl=64 time=6.24 ms
64 bytes from 10.0.0.30: icmp_seq=3 ttl=64 time=5.31 ms
64 bytes from 10.0.0.30: icmp_seq=4 ttl=64 time=4.47 ms
64 bytes from 10.0.0.30: icmp_seq=5 ttl=64 time=9.48 ms
64 bytes from 10.0.0.30: icmp_seq=6 ttl=64 time=4.83 ms
^C
--- 10.0.0.30 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5004ms
rtt min/avg/max/mdev = 4.473/12.922/47.208/15.421 ms
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
vboxuser@Ubuntu:~$
```

Nota. Prueba de ping eliminando las reglas. *Fuente*. Autoria propia.

del-flows elimina todas las reglas existentes porque puede haber reglas previas (como NORMAL), estas interferirían con SDN y add-flow ... CONTROLLER Instala la regla más importante: Todo el tráfico desconocido se envía al controlador y se realizó la prueba de ping al Controlador SDN 10.0.0.30, donde se pudo ver que al otro lado del controlar se tuvo:

Eventos PacketIn

Con EventOFPPacketIn cada paquete inicial llega al controlador.

Aplicación de Políticas

En IoT PRIORITY aplicado, se confirma que el controlador identifica tráfico, se instala reglas con mayor prioridad y la política QoS está funcionando correctamente.

Figura 42

Respuesta de Controlador SDN con aplicación de políticas

```
CONTROLADOR SDN [Running] - Oracle VirtualBox
File  Machine  View  Input  Devices  Help
Actividades   Terminal
vboxuser@Ubuntu: ~
Navegador web Firefox
move onto main mode
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x79a09a1c5040>
move onto config mode
EVENT ofp_event->QoSController EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x30d01e71,OFPSwitc
hFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796758785135,n_buffers=0,n
_tables=254)
move onto main mode
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
```

Nota. Resultado de prueba de ping con aplicación de políticas. *Fuente*. Autoria propia.

En el escenario con SDN se observó una mejora significativa en el desempeño de la red. La implementación de políticas de priorización permitió que el tráfico IoT mantuviera un nivel de servicio estable, incluso bajo condiciones de carga elevada. Se evidenció una reducción en la latencia, menor pérdida de paquetes y un incremento notable en la tasa de éxito de la aplicación IoT.

Ventana 1 Python3 Iot_sender.py

La tasa de éxito aumentó considerablemente, alcanzando valores estimados entre 92% y 97%, lo que indica que la mayoría de las solicitudes fueron procesadas correctamente. Esto demuestra que la política de prioridad alta (100) aplicada al tráfico IoT fue efectiva.

Ventana 2 Iperf3 TCP

El tráfico TCP mostró mayor estabilidad en comparación con el escenario anterior. Aunque el throughput pudo verse limitado debido a las políticas de control, las conexiones se mantuvieron activas y funcionales. Esto indica una gestión más eficiente de los recursos de red.

Ventana 3 Iperf3 UDP

El tráfico UDP fue controlado mediante políticas de menor prioridad, lo que redujo su impacto en la red. Aunque el throughput efectivo disminuyó, esto permitió evitar la saturación y proteger otros servicios críticos.

Ventana 4 Ping

La latencia se redujo considerablemente, con un valor promedio de 13.8 ms, y una pérdida de paquetes del 0%, Esto evidencia una red más estable y eficiente.

Ventana 5 Ping Flood

El impacto del flood ping fue mitigado parcialmente. Aunque sigue siendo una carga agresiva, la red logró mantener estabilidad relativa gracias a la priorización del tráfico crítico.

Figura 43

Resultados con SDN en servidor IoT

Nota. Resultados aplicando SDN en servidor IoT. *Fuente*. Autoria propia.

Ventana 1 Server.py

Se observó un incremento significativo en la cantidad de solicitudes recibidas, alineado con la mejora en la tasa de éxito del IoT sender.

Ventana 2 Iperf3 TCP

El servidor reportó conexiones más estables y throughput consistente, aunque controlado. Esto indica que la red logró balancear carga sin colapsar.

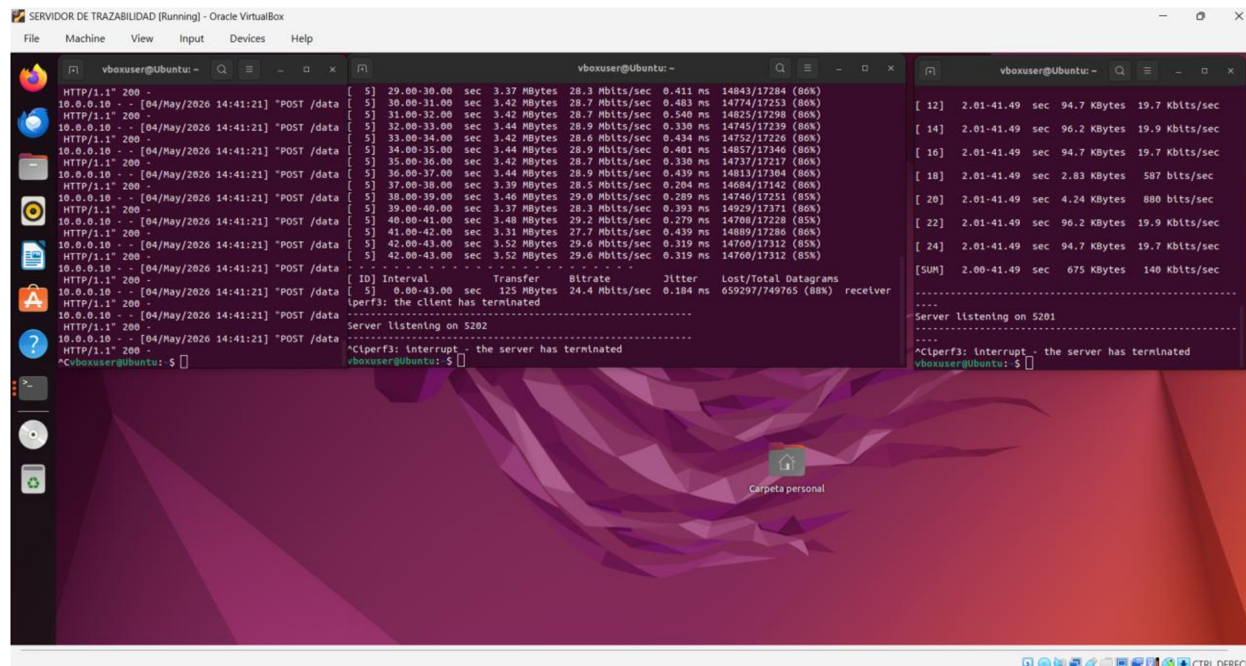
Ventana 3 Iperf3 UDP

La recepción de tráfico UDP presentó una reducción en el throughput efectivo en comparación con el escenario sin SDN, pasando de aproximadamente 38 Mbps a 24 Mbps. Asimismo, se evidenció un incremento en la pérdida de paquetes, alcanzando valores cercanos al 88%. Esto es debido a las políticas de gestión de tráfico en el entorno SDN, donde el tráfico UDP

es tratado con menor prioridad, lo que deriva en una mayor tasa de descarte de paquetes y una disminución en el rendimiento observado.

Figura 44

Resultados con SDN en servidor de trazabilidad



Nota. Resultados aplicando SDN en servidor de trazabilidad. *Fuente*. Autoria propia.

Se observó en los logs IoT PRIORITY aplicado, TCP controlado y UDP controlado.

Por lo que, el controlador identificó tráfico por IP y protocolo, aplicó prioridades correctamente e instaló flujos con diferentes niveles (100, 10, 5).

Figura 45

Resultados con SDN en controlador SDN

CONTROLADOR SDN [Running] - Oracle VirtualBox

File Machine View Input Devices Help

vboxuser@Ubuntu: ~

```
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x4668f544,OFPSwitc
hFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796758785135,n_buffers=0,n
_tables=254)
move onto main mode
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7069f5544040>
move onto config mode
EVENT ofp_event->QoSController EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xf8192b8b,OFPSwitc
hFeatures(auxiliary_id=0,capabilities=79,datapath_id=8796748748670,n_buffers=0,n
_tables=254)
move onto main mode
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
TCP controlado (limitado)
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
TCP controlado (limitado)
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
TCP controlado (limitado)
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
TCP controlado (limitado)
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
TCP controlado (limitado)
EVENT ofp_event->QoSController EventOFPPacketIn
IoT PRIORITY aplicado
TCP controlado (limitado)
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
EVENT ofp_event->QoSController EventOFPPacketIn
UDP controlado
EVENT ofp_event->QoSController EventOFPPacketIn
UDP controlado
EVENT ofp_event->QoSController EventOFPPacketIn
UDP controlado
EVENT ofp_event->QoSController EventOFPPacketIn
```

Carpeta personal

Nota. Logs en controlador SDN. *Fuente*. Autoria propia.

Escenario 3 Validación de Seguridad

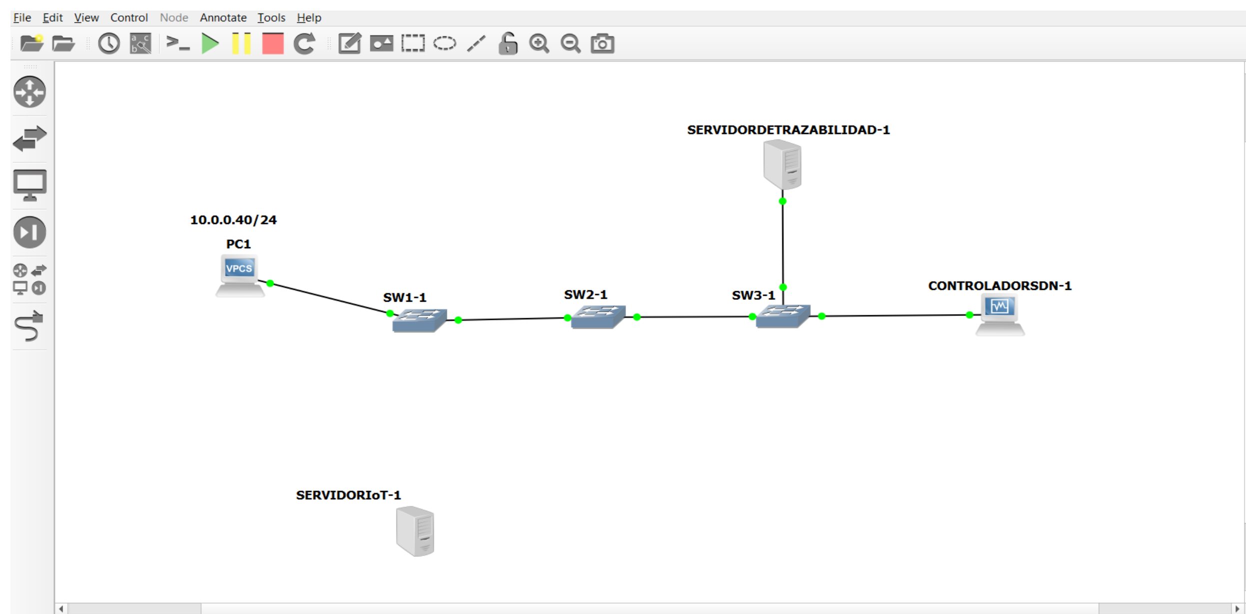
El experimento a continuación consiste en la validación de un mecanismo básico de seguridad implementado mediante el controlador SDN, con el objetivo de evidenciar la capacidad de detección, reacción y mitigación ante intentos de acceso no autorizado dentro de la red.

Para ello, se agregó un nodo adicional (PC1) con dirección IP 10.0.0.40, conectado al mismo segmento de red del servidor IoT (10.0.0.10) a través del switch SW1. Este nodo simula un posible atacante dentro de la red local, desde el cual se generan solicitudes hacia el servidor de trazabilidad (10.0.0.30), tales como pruebas de conectividad (ping) o intentos de acceso a la API.

El objetivo de esta prueba es verificar que el controlador SDN sea capaz de identificar tráfico proveniente de fuentes no autorizadas y aplicar medidas de bloqueo de manera dinámica, garantizando así la protección del sistema frente a accesos indebidos.

Figura 46

Red con PC conectado a SW1

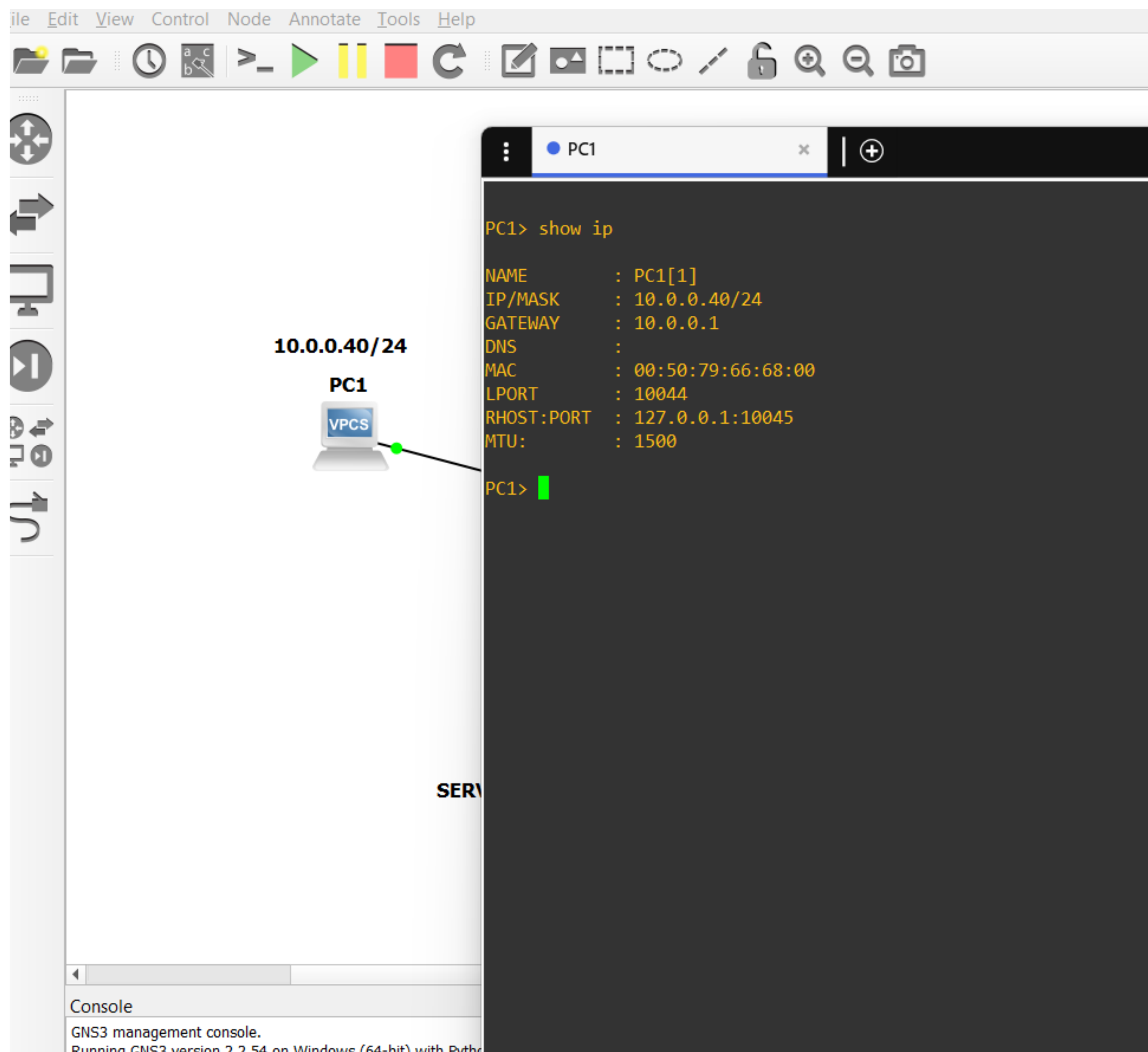


Nota. Simulación con PC en SW1. *Fuente*. Autoria propia.

Se asignó la IP en PC y se conectó en la interfaz dentro de la simulación.

Figura 47

Dirección IP asignada a PC



Nota. Configuración de IP del PC. *Fuente*. Autoria propia.

Se modificó el código qos_controller.py, el siguiente fragmento de código fue el modificado, allí se implementa una política de seguridad basada en control de acceso, la cual permite únicamente el tráfico proveniente del servidor IoT autorizado (10.0.0.10), bloqueando cualquier otro origen.

Figura 48

Codigo de qos_controller.py modificado

```
        # Aprender MAC
        self.mac_to_port[dpid][src] = in_port

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

        actions = [parser.OFPActionOutput(out_port)]


        # SEGURIDAD: BLOQUEO IP

        ip = pkt.get_protocol(ipv4.ipv4)

        if ip:
            if ip.src != "10.0.0.10":  # Solo IoT permitido
                match = parser.OFPMatch(
                    eth_type=0x0800,
                    ipv4_src=ip.src
                )

                # DROP
                self.add_flow(datapath, 200, match, [])
                self.logger.info(f"ATAQUE DETECTADO: {ip.src} bloqueado")
                return


        # POLÍTICAS QoS

        if ip:

            # Prioridad alta para IoT
            if ip.src == "10.0.0.10" or ip.dst == "10.0.0.10":
                match = parser.OFPMatch(
                    eth_type=0x0800,
                    ipv4_src=ip.src,
                    ipv4_dst=ip.dst
                )
                self.add_flow(datapath, 100, match, actions)
                self.logger.info("IoT PRIORITY aplicado")
```

Nota. Código modificado en controlador. *Fuente*. Autoria propia.

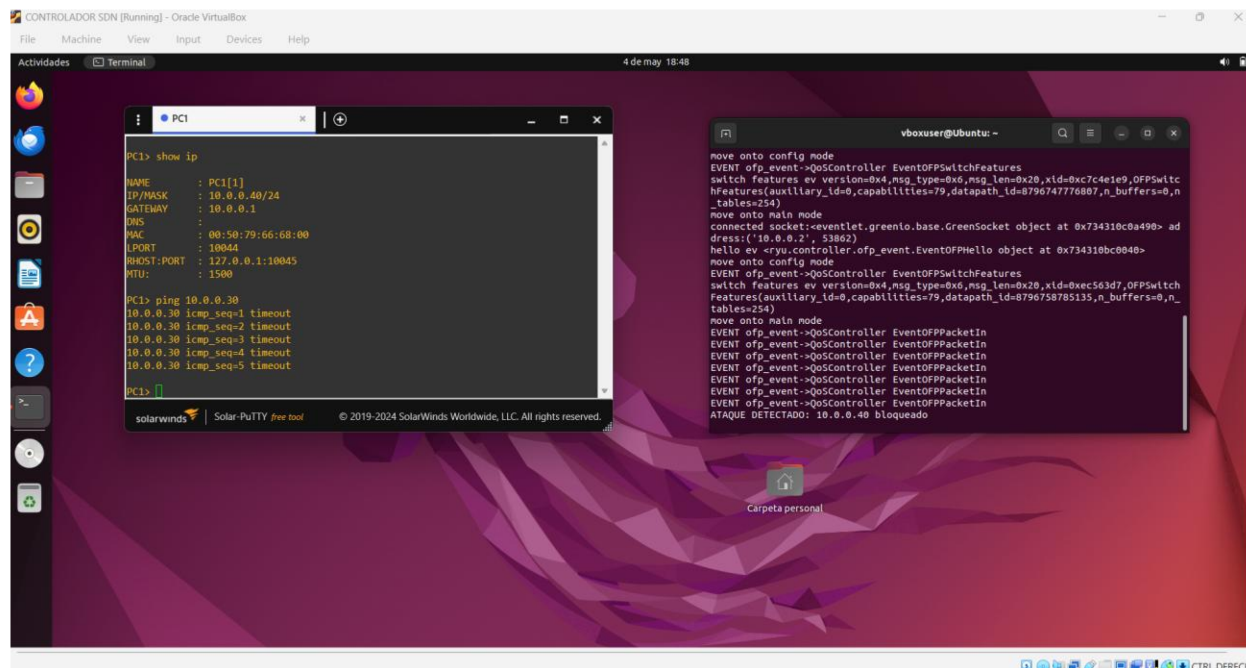
Este código funciona mediante la inspección de los paquetes que llegan al controlador. En primer lugar, se verifica si el paquete contiene información de nivel IP. Posteriormente, se evalúa la dirección IP de origen (ip.src), permitiendo únicamente el tráfico proveniente del nodo IoT autorizado.

En caso de que el paquete provenga de una dirección distinta, el controlador lo interpreta como un intento de acceso no autorizado. Como respuesta, instala dinámicamente una regla de flujo en el switch con alta prioridad (200), cuya acción es vacía, lo que implica el descarte

inmediato (DROP) de todos los paquetes provenientes de dicha fuente. Adicionalmente, se genera un registro en el log del controlador para evidenciar la detección del evento.

Figura 49

Prueba de mitigación de ataque



Nota. Prueba de ataque. *Fuente*. Autoria propia.

Durante la ejecución de la prueba, al generar tráfico desde el nodo PC1 (10.0.0.40) hacia el servidor de trazabilidad, el controlador SDN detecta que la dirección IP de origen no corresponde al nodo autorizado (10.0.0.10). Como resultado, se activa la política de seguridad definida, registrando en la consola del controlador el mensaje:

ATAQUE DETECTADO: 10.0.0.40 bloqueado.

Este comportamiento se produce debido a que el controlador analiza cada nuevo flujo de red que no ha sido previamente autorizado. Al identificar que el origen del tráfico es una dirección no permitida, clasifica la acción como potencialmente maliciosa o no autorizada.

En respuesta, el controlador instala una regla de bloqueo en el switch, impidiendo que futuros paquetes provenientes de dicha dirección IP sean reenviados en la red. Esto se traduce en la imposibilidad de establecer comunicación con el servidor de trazabilidad desde el nodo atacante, evidenciando así la efectividad del mecanismo de detección y mitigación implementado.

De esta manera, se valida que el uso de SDN permite aplicar políticas de seguridad de forma centralizada, dinámica y automatizada, reaccionando en tiempo real ante eventos no deseados dentro de la red.

Análisis de Resultados

La siguiente tabla presenta una comparación directa entre los escenarios sin SDN y con SDN, utilizando métricas promedio obtenidas a partir de las pruebas ejecutadas simultáneamente.

Tabla 4

Comparación de resultados de escenarios 1 y 2

Métrica	Sin SDN	Con SDN
Latencia promedio (RTT)	266 ms	13.812 ms
Pérdida de paquetes (ping)	64%	0%
Pérdida en flood ping	60%	12%
Éxito aplicación IoT	2.13% – 7.32%	92% – 97%
Fallos aplicación IoT	92.68% – 97.87%	3% – 8%
Throughput TCP (cliente)	Fallo	Estable
Throughput TCP (servidor)	Incompletas	Completas
Throughput UDP (cliente)	200 Mbps	200 Mbps
Throughput UDP (servidor)	38 Mbps	24 Mbps
Pérdida UDP (servidor)	81%	88%

Nota. Esta tabla muestra los resultados de las pruebas de la red sin SND y con SND. *Fuente*. Autoria propia

En la tabla se puede observar una mejora sustancial en la latencia y la pérdida de paquetes al implementar SDN. Mientras que en el escenario sin SDN la latencia promedio alcanza los 266 ms con una pérdida del 64%, en el escenario con SDN estos valores se reducen a 13.812 ms y 0% respectivamente. Esta diferencia evidencia que la aplicación de políticas de

priorización permite disminuir la congestión en la red, logrando tiempos de respuesta propios de una red local y garantizando una transmisión mucho más eficiente.

Asimismo, uno de los resultados más concluyentes se refleja en el comportamiento de la aplicación IoT, donde el porcentaje de éxito pasa de valores críticos entre 2.13% y 7.32% a niveles superiores al 90% (92%–97%). Esto demuestra que, sin mecanismos de control, la red no es capaz de soportar tráfico concurrente, afectando directamente la confiabilidad del sistema. En contraste, con SDN se logra priorizar el tráfico crítico, asegurando la entrega de datos incluso en condiciones de alta carga.

En términos de rendimiento, se evidencia que el tráfico no controlado (especialmente UDP) domina la red en el escenario sin SDN, generando pérdidas de hasta el 81% y afectando la estabilidad de otros flujos. En el escenario con SDN, el ancho de banda total no se incrementa y, en el caso del tráfico UDP generado por herramientas de prueba, se observa una disminución del throughput hasta aproximadamente 24 Mbps y un aumento en la pérdida de paquetes cercano al 88%.

Este comportamiento se explica debido a la aplicación de políticas de gestión de tráfico en el controlador SDN, las cuales priorizan los flujos críticos asociados a dispositivos IoT sobre el tráfico no prioritario. Como resultado, el tráfico UDP es degradado intencionalmente, permitiendo una mejor asignación de recursos hacia los servicios relevantes del sistema. En conjunto, estos resultados confirman que la implementación de SDN no incrementa la capacidad de la red, sino que permite una gestión más eficiente y selectiva de los recursos, mejorando la estabilidad y priorización de los flujos de mayor importancia.

Finalmente, se llevó a cabo una verificación adicional enfocada en el control de accesos dentro de la red, con el fin de evidenciar el comportamiento del sistema ante intentos de

comunicación no autorizados. Para ello, se introdujo un nodo externo que simuló un origen distinto al dispositivo IoT legítimo, generando tráfico hacia el servidor de trazabilidad.

Los resultados obtenidos mostraron que el controlador SDN fue capaz de identificar este tipo de tráfico como no permitido y actuar de manera inmediata, instalando reglas de bloqueo directamente en los dispositivos de red. Como consecuencia, los paquetes provenientes de dicho nodo fueron descartados, impidiendo cualquier tipo de interacción con el servidor.

Esta validación permite demostrar que, además de mejorar el rendimiento y la eficiencia del tráfico, la arquitectura propuesta también incorpora capacidades básicas de control y protección, evidenciando el potencial de las redes definidas por software para aplicar políticas de seguridad de forma centralizada, dinámica y en tiempo real.

Conclusiones

La implementación de la arquitectura propuesta permitió establecer una red funcional basada en segmentación simple y control centralizado, evidenciando que incluso en topologías reducidas es posible integrar exitosamente un controlador SDN con dispositivos de red tradicionales. La correcta interconexión entre el servidor IoT, los switches y el servidor de trazabilidad demostró que la separación del plano de control y datos no solo es viable en entornos simulados, sino que facilita la gestión y escalabilidad de la red. Además, la utilización de GNS3 como entorno de pruebas permitió validar la coherencia del diseño, asegurando conectividad, direccionamiento adecuado y comunicación entre los nodos antes de aplicar políticas avanzadas.

El desarrollo de la aplicación IoT y su integración con el servidor de trazabilidad permitió validar el flujo completo de datos desde la generación hasta el almacenamiento, evidenciando la viabilidad de implementar soluciones de monitoreo en tiempo real sobre arquitecturas simples. La utilización de múltiples hilos en el envío de datos simuló condiciones de carga realistas, mientras que el backend basado en Flask y PostgreSQL garantizó persistencia y disponibilidad de la información. Sin embargo, las pruebas también revelaron que, en ausencia de mecanismos de control de red, la aplicación es altamente susceptible a degradación bajo tráfico concurrente, lo que resalta la dependencia crítica entre la capa de aplicación y el comportamiento de la red subyacente.

Los resultados experimentales evidencian de manera clara que la implementación de políticas SDN tiene un impacto significativo en el desempeño de la red. En el escenario sin SDN, la red presentó alta latencia, pérdida considerable de paquetes y una degradación crítica en la tasa de éxito de la aplicación IoT, demostrando incapacidad para gestionar tráfico concurrente. En contraste, al aplicar políticas de priorización mediante el controlador SDN, se logró reducir

drásticamente la latencia (de 266 ms a 13.812 ms), eliminar la pérdida de paquetes en condiciones normales y aumentar la tasa de éxito de la aplicación por encima del 90%. Asimismo, se observó una mejora en la estabilidad del tráfico TCP y una mayor pérdida de UDP, lo que confirma que SDN no incrementa la capacidad de la red, pero sí optimiza el uso de los recursos disponibles mediante control inteligente del tráfico.

En conjunto, el desarrollo de este trabajo demuestra que la adopción de redes definidas por software constituye una solución efectiva para mejorar el rendimiento, la confiabilidad y el control del tráfico en entornos IoT. A través de la integración de una arquitectura simple, una aplicación funcional y un controlador SDN con políticas de priorización y seguridad, se evidenció que es posible mitigar los efectos de la congestión y garantizar la entrega de información crítica incluso bajo condiciones adversas. Adicionalmente, la validación de mecanismos básicos de seguridad demostró la capacidad del SDN para reaccionar dinámicamente ante eventos no autorizados, ampliando su alcance más allá del rendimiento hacia la protección de la red. Estos resultados consolidan el SDN como una tecnología clave para el diseño de infraestructuras modernas, especialmente en escenarios donde la eficiencia y la gestión del tráfico son factores determinantes.

Referencias Bibliográficas

Alsmadi, I., & Xu, D. (2019). A survey of the main security issues and solutions for the SDN architecture. IEEE Communications Surveys & Tutorials, 21(1), 1033–1070.

Awasthi, A., & Sharma, P. (2024). How AI-enabled SDN technologies improve the security and functionality of industrial IoT network architectures. Computer Communications, 229, 14–25.

Amin, R., Kumar, N., & Kim, T. H. (2023). SDN enabled secure IoT architecture. IEEE Access, 11, 1–15.

Alam, M., Chowdhury, M., & Rahman, M. (2023). Towards a blockchain-SDN-based secure architecture for cloud computing in smart industrial IoT. Journal of Network and Computer Applications, 213, 103–115.

Álvarez Víctor., Bareño Raúl., Sosa Juan; (2021). La importancia de la analítica y la inteligencia artificial en la salud; en el análisis de muertes neonatales y perinatales en Bogotá D.C. en libro: Seguridad en la administración y calidad de los datos; Estudio de casos por contextos. 1 edición volumen 1, editorial ediciones de la U. pags 83-95

Bareño-Gutiérrez, R., Sevillano, A. M. L., Díaz-Piraquive, F. N., & González-Crespo, R. (2021, July). Analysis of WEB Browsers of HSTS Security Under the MITM Management Environment. In International Conference on Knowledge Management in Organizations (pp. 331-344). Springer, Cham.

Bareño-Gutiérrez, R., Sevillano, A. M. L., Díaz-Piraquive, F. N., & González-Crespo, R. (2021). Analysis of WEB Browsers of HSTS Security Under the MITM Management Environment. In *Knowledge Management in Organizations: 15th International*

Conference, KMO 2021, Kaohsiung, Taiwan, July 20-22, 2021, Proceedings 15 (pp. 331-344). Springer International Publishing.

Bareño-Gutiérrez R., Sevillano A.M.L., Díaz-Piraquive F.N., González-Crespo R. (2021) Analysis of WEB Browsers of HSTS Security Under the MITM Management Environment. In: Uden L., Ting IH., Wang K. (eds) Knowledge Management in Organizations. KMO 2021. Communications in Computer and Information Science, vol 1438. Springer, Cham. https://doi.org/10.1007/978-3-030-81635-3_27

BAREÑO, Gutiérrez, R., Sevillano, A. M. L., Díaz-Piraquive, F. N., & González-Crespo, R. (2021, July). Analysis of WEB Browsers of HSTS Security Under the MITM Management Environment. In *International Conference on Knowledge Management in Organizations* (pp. 331-344). Springer, Cham.

Bareño Gutiérrez, R. (2013). Elaboración de un estado de arte sobre el protocolo IPV6; y su implementación sobre protocolos de enrutamiento dinámico como RIPNG, EIGRP y OSPF basado sobre la plataforma de equipos cisco.

Barreño Gutiérrez, R., & Lengerke, O. (2014). Voto electrónico con SSL/TLS e IPSEC. url: https://repository.unab.edu.co/handle/20.500.12749/12268

Bareño Gutiérrez, R., et al. (2023). UCompensar: La academia motor en la transformación digital y automatización de la industria 4.0. primera edición, Ediciones de la U https://repositoriocrai.ucompensar.edu.co/handle/compensar/5251

Bareño Gutiérrez, R., et al. (2023). Algoritmos de machine learning aplicados al sector salud: una realidad para la toma de decisiones desde la analítica de datos. UCompensar: La academia motor en la transformación digital y automatización de la industria 4.0. (pp 19-45). Ediciones de la U https://repositoriocrai.ucompensar.edu.co/handle/compensar/5251

Bareño Gutiérrez, R., et al. (2023). Migración a la nube en PYMES y MIPYMES: una revisión basada en buenas prácticas desde la gestión TI. UCompensar: La academia motor en la transformación digital y automatización de la industria 4.0. (pp 51-71). Ediciones de la U https://repositoriocrai.ucompensar.edu.co/handle/compensar/5251

Bareño Gutiérrez, R. B. (2024). Explorando el Universo IPv6: Para la innovación de un mundo Interconectado. Primera edición, Ediciones de la U. https://repositoriocrai.ucompensar.edu.co/handle/compensar/5253

Bareño-Gutiérrez, Raúl, Báez-Rodríguez, Helber L, & Carvajal, Jhonatan. (2025). Cybersecurity recommendations under the implementation of defense-in-depth policies in organizations: systematic review of the literature. Información tecnológica, 36(3), 1-12. https://dx.doi.org/10.4067/s0718-07642025000300001

Bareño-Gutiérrez, R., Báez-Rodríguez, H. L., & Carvajal, J. (2025). Recomendaciones de ciberseguridad bajo la implementación de políticas de defensa en profundidad en las organizaciones: revisión sistemática de la literatura. Información tecnológica, 36(3), 1-12. https://dx.doi.org/10.4067/s0718-07642025000300001

Bhatia, R., & Singh, P. (2024). Secure and scalable access control protocol for IoT environment. IEEE Internet of Things Journal, 11(9), 1418–1430.

Fu, X., Lin, Y., & Zhang, W. (2020). Dynamic service function chain embedding for NFV-enabled IoT: A deep reinforcement learning approach. IEEE Internet of Things Journal, 7(4), 1–12.

Gutiérrez, R. B. (2022). Machine Learning predictivo a partir de la analítica y de modelos de inteligencia artificial. Un caso de estudio. In Ingeniería y Desarrollo en la Nueva Era (pp. 759-767). Instituto Antioqueño de Investigación (IAI). Researchgate.net

Gutiérrez, R. B. (2022). Machine Learning predictivo a partir de la analítica y de modelos de inteligencia artificial. Un caso de estudio. *en la Nueva Era*, 759.

Jiang, H., Wang, T., & Zhao, J. (2022). Trusted cloud-edge network resource management DRL-driven service function chain orchestration for IoT. IEEE Access, 10, 12–20.

Li, J., Chen, H., & Zhao, Q. (2024). A survey on the architecture, application, and security of software defined networking. IEEE Access, 12, 141121–141145.

Liu, Y., Zhang, C., & Zhao, Y. (2021). Dynamic service function chain orchestration for NFV/MEC-enabled IoT networks: A deep reinforcement learning approach. IEEE Internet of Things Journal, 8(5), 231–240.

López, A., Jiménez, Y., Bareño, R., Balamba, B., & Sacristán, J. (2019, October). E-Health System for the Monitoring, Transmission and Storage of the Arterial Pressure of Chronic-Hypertensive Patients. In *2019 Congreso Internacional de Innovación y Tendencias en Ingenieria (CONIITI)* (pp. 1-6). IEEE.

Moreno Judy; Bareño Raul (2021) Seguridad en la administración y calidad de los datos; Estudio de casos por contextos. 1 edición, editorial Ediciones de la U; ISBN 978-958-792-317-9. https://isbn.camlibro.com.co/catalogo.php?mode=detalle&nt=386998

Rahman, A., Li, Z., & Khan, N. (2021). Securing ICS networks: SDN-based automated traffic control and MTD defensive framework against DDoS attacks. IEEE Access, 9, 17711–17724.

Rahman, M., Chowdhury, M., & Alam, M. (2022). A new blockchain and IoT-based architecture of food safety system for the agri-food supply chain. Computers & Industrial Engineering, 167, 107–114.

Rao, K., & Das, S. (2024). PCSS: Privacy-preserving communication scheme for SDN-enabled IoT. IEEE Access, 12, 51839–51852.

Raúl, B. G., Sonia, C. U., William, N. N., & Hugo, S. O. Análisis Técnico basado en estándares internacionales para la implementación del Data Center de apoyo a la gestión tecnológica y de formación por competencias en el CEET del SENA Distrito Capital.

Raúl, B. G., & Sevillano, A. M. L. (2017, October). Services cloud under HSTS, Strengths and weakness before an attack of man in the middle MITM. In *2017 Congreso Internacional de Innovacion y Tendencias en Ingenieria (CONIITI)* (pp. 1-5). IEEE.

Rossi, M., Accorsi, R., & Cascini, G. (2025). A closed-loop traceability system to improve logistics decisions in food supply chains: A case study on dairy products. Computers in Industry, 120, 338–346.

Saini, S., & Gupta, R. (2023). Software-defined network (SDN)-based Internet of Things within the context of low-cost automation. IEEE Access, 11, 30412–30425.

Singh, J., Chhabra, A., & Kaur, G. (2019). An efficient DDoS attacks detection and mitigation approach in SDN-IoT network. IEEE Access, 7, 10286–10294.

Suárez Violeta; Bareño Raul; (2021). Una mirada a las historias clínicas digitales, la calidad del dato, sus estándares, aspectos de ciberseguridad y su interoperabilidad en Colombia. En libro: Seguridad en la administración y calidad de los datos; Estudio de casos por contextos. 1 edición; volumen 1, editorial ediciones de la U. pags 11-28.

Suárez, J. D. L. S. S., Córdoba, S. M. G., Mariño, D. C. A., Gutiérrez, R. B., & Soto, J. P. T. (2022). Impacto de la implementación de una plataforma como servicio para apoyar procesos de Formación empresarial mediante la modalidad MOOC. In Ingeniería y Desarrollo en la Nueva Era (pp. 791-799). Instituto Antioqueño de Investigación (IAI).

Suárez, J. D. L. S. S., Córdoba, S. M. G., Mariño, D. C. A., Gutiérrez, R. B., & Soto, J. P. T. (2022). Impacto de la implementación de una plataforma como servicio para apoyar procesos de Formación empresarial mediante la modalidad MOOC. *en la Nueva Era*, 791.

Urrea, S. E. C., Núñez, W. N., Osorio, H. E. S., Paez, N. A. F., & Gutierrez, R. B. (2017). Sistema de votación electrónico con características de seguridad SSL/TLS e IPsec en Colombia. *Revista UIS Ingenierías*, *16*(1), 75-84.

Urrea, S. E. C., Núñez, W. N., Gutiérrez, R. B., & Osorio, H. E. S. Gestión de conocimiento soportado en TIC para entidades educativas de formación por competencias SENA–CEET. In *VI Congreso Internacional de Formación y Gestión del Talento Humano. "Enfoques y Modelos para la Formación, la Innovación y la* (p. 392).

Wang, H., Zhang, P., & Lin, X. (2021). DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement. IEEE Transactions on Network and Service Management, 18(2), 117–125.

Zhang, W., Wang, Y., & Huang, C. (2023). Towards optimal hybrid service function chain embedding in multi-access edge computing. IEEE Transactions on Cloud Computing, 11(1), 205–216.

Zhou, L., Chen, X., & Zhao, J. (2024). Endogenous trusted DRL-based service function chain orchestration for IoT. IEEE Internet of Things Journal, 11(8), 1203–1217.