UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CARRERA DE TELECOMUNICACIONES

Desarrollo de un modelo de seguridad para redes SDN basado en machine learning empleando herramientas Open Source

Trabajo de Titulación para optar al título de Ingeniero en Telecomunicaciones

Autor:
Enriquez Jaramillo, Kevin Walter

Tutor:
Phd. Pedro Fernando Escudero Villa.

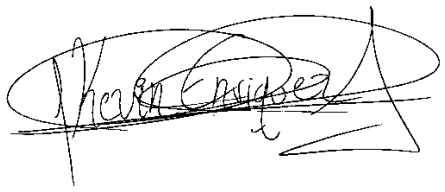
Riobamba, Ecuador. 2025

DECLARATORIA DE AUTORÍA

Yo, Kevin Walter Enriquez Jaramillo, con cédula de ciudadanía 1724631641, autor del trabajo de investigación titulado: Desarrollo de un modelo de seguridad para redes sdn basado en machine learning empleando herramientas open source, certifico que la producción, ideas, opiniones, criterios, contenidos y conclusiones expuestas son de mí exclusiva responsabilidad.

Asimismo, cedo a la Universidad Nacional de Chimborazo, en forma no exclusiva, los derechos para su uso, comunicación pública, distribución, divulgación y/o reproducción total o parcial, por medio físico o digital; en esta cesión se entiende que el cesionario no podrá obtener beneficios económicos. La posible reclamación de terceros respecto de los derechos de autor (a) de la obra referida, será de mi entera responsabilidad; librando a la Universidad Nacional de Chimborazo de posibles obligaciones.

En Riobamba, 1 de mayo de 2025.



Kevin Walter Enriquez Jaramillo
C.I: 1724631641

DICTAMEN FAVORABLE DEL PROFESOR TUTOR

Quien suscribe, **Pedro Fernando Escudero Villa** catedrático adscrito a la Facultad de Ingeniería, por medio del presente documento certifico haber asesorado y revisado el desarrollo del trabajo de investigación **"Desarrollo de un modelo de seguridad para redes sdn basado en machine learning empleando herramientas open source"**, bajo la autoría de **Kevin Walter Enriquez Jaramillo**; por lo que se autoriza ejecutar los trámites legales para su sustentación.

Es todo cuanto informar en honor a la verdad; en Riobamba, a los 1 días del mes de mayo del 2025

Firmado electrónicamente por:
PEDRO FERNANDO ESCUDERO VILLA
Validar únicamente con FirmaEC

_____

PhD. Pedro Fernando Escudero Villa
C.I: 0603612524
TUTOR

CERTIFICADO DE LOS MIEMBROS DEL TRIBUNAL

Quienes suscribimos, catedráticos designados Miembros del Tribunal de Grado para la evaluación del trabajo de investigación Desarrollo de un modelo de seguridad para redes sdn basado en machine learning empleando herramientas open source por Kevin Walter Enriquez Jaramillo, con cédula de identidad número 1724631641, bajo la tutoría de PhD. Pedro Fernando Escudero Villa; certificamos que recomendamos la APROBACIÓN de este con fines de titulación. Previamente se ha evaluado el trabajo de investigación y escuchada la sustentación por parte de su autor; no teniendo más nada que observar.
De conformidad a la normativa aplicable firmamos, en Riobamba 26 de mayo del 2025.

PhD. Daniel Santillán
PRESIDENTE DEL TRIBUNAL DE GRADO

PhD. Antonio Meneses
MIEMBRO DEL TRIBUNAL DE GRADO

Mgs. Giovanny Cuzco
MIEMBRO DEL TRIBUNAL DE GRADO

C E R T I F I C A C I Ó N

Que, **Enriquez Jaramillo Kevin Walter** con CC: **1724631641**, estudiante de la Carrera **TELECOMUNICACIONES**, Facultad de **INGENIERÍA**; ha trabajado bajo mi tutoría el trabajo de investigación titulado **"DESARROLLO DE UN MODELO DE SEGURIDAD PARA REDES SDN BASADO EN MACHINE LEARNING EMPLEANDO HERRAMIENTAS OPEN SOURCE**", cumple con el 4 %, de acuerdo al reporte del sistema Anti plagio **COMPILATIO**, porcentaje aceptado de acuerdo a la reglamentación institucional, por consiguiente, autorizo continuar con el proceso.

Riobamba, 06 de mayo de 2025

Firmado electrónicamente por:
PEDRO FERNANDO ESCUDERO VILLA
Validar únicamente con FirmaEC

PhD. Pedro Fernando Escudero Villa
TUTOR

DEDICATORIA

Dedico este presente proyecto de investigación a Dios y a mi familia, porque son ellos los que me han apoyado de forma incondicional en el trayecto de mi vida y han creído siempre en mi persona, a mi madre Doris Marizol Jaramillo y en el cielo a mi padre Walter Wilfrido Enriquez, por ser mis más grandes mentores al enseñarme sobre la grandeza que tiene el ser humano, a mi tía Lucia Jaramillo por ser como mi segunda madre, a mis primas Emilia, Erika y Suleny, por ser como mis hermanas. A mi abuela Fanny, que vive en mi memoria, por cuidar de mí en mis etapas de niñez y adolescencia. También mencionó a cada una de esas personas que amé y fueron parte de esta etapa de mi vida.

Dedico esto a mis profesores, aquellos hombres y mujeres que con su dedicación y compromiso al compartir sus experiencias y conocimientos en el campo de la telecomunicación, al fomentar la curiosidad y las ganas por entender sobre las innovaciones tecnológicas. Dedico esto a todos mis amigos, en especial a Alejandro Cueva y Darwin Armijos porque siempre me acompañaron, con horas de estudio, risas, desvelos y el apoyo mutuo hasta lograr culminar cada semestre.

AGRADECIMIENTO

Primeramente, quiero agradecer a Dios porque me ha permitido alcanzar mis logros y metas, a mi tutor de tesis, PhD Pedro Escudero, por confiar en este proyecto, guiarme con paciencia y sabiduría en el desarrollo de este proceso y motivar me ha ver más allá de mis propios límites.

Agradecer a mi familia, por el inmenso esfuerzo que han realizado para ayudarme a continuar con mis estudios, afrontando muchos problemas como malas personas, gente injusta, personas que quise ya no son parte de mi vida, la competencia desleal. Permitiendo motivarme, superarme y demostrar que puedo llegar a hacer grandes cosas después de todo. La familia, gracias por estar siempre a mi lado, sosteniéndome, guiándome en los momentos más difíciles, demostrándome que a pesar de todo soy alguien especial y excepcional.

Kevin Walter Enriquez Jaramillo

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En la era digital actual, los constantes cambios tecnológicos y el incremento de usuarios, así como la aparición de nuevas plataformas de servicios, han evidenciado las limitaciones de escalabilidad en relación con el espacio en las redes tradicionales. Ante esta problemática, han surgido las redes definidas por software (SDN), que proponen una arquitectura centralizada y programable. Este nuevo tipo de redes ofrece muchas ventajas, como una gestión eficiente y una mayor flexibilidad en el manejo de recursos, pero también plantea retos importantes en el ámbito de la seguridad. El presente proyecto de investigación se relaciona con la problemática de las vulnerabilidades en este tipo de topologías de red. El vector de ataque a tratar, presente también en redes convencionales, es el denominan MAC Spoofing, que tiene impacto en la capa de enlace de datos. Su objetivo es la interceptación o apropiación de datos pertenecientes a dispositivos legítimos, generando robo de información y suplantación de identidad dentro de la red SDN. Como desarrollo de una contramedida, se plantea un modelo de seguridad que permita predecir estos vectores de ataque mediante la implementación de un algoritmo de clasificación denominado Random Forest. Para la creación de la topología de red personalizada en el entorno de pruebas se emplea la herramienta Mininet, y para el componente de control se utilizará RYU, tanto en la monitorización del tráfico como en la implementación del prototipo de seguridad mediante la creación de una aplicación. Se utilizo un entorno virtualizado como plataforma de pruebas para evaluar la precisión y eficiencia de los modelos generados frente a ataque simulados.

Palabras claves: Mininet, RYU, aprendizaje supervisado, redes SDN, ciberseguridad.

ABSTRACT

In today's digital age, constant technological changes, the increasing number of users, and the rise of new service platforms have revealed scalability limitations in traditional network architectures. To address these issues, Software-Defined Networking (SDN) has emerged, introducing a centralized and programmable network structure. SDN provides significant advantages, including efficient resource management and enhanced flexibility. However, it also presents new and complex security challenges. This research project focuses on security vulnerabilities within SDN environments. One specific attack vector studied is MAC Spoofing, which also affects traditional networks and targets the data link layer. The attack's goal is to intercept or impersonate legitimate devices in order to steal sensitive information and perform identity spoofing within the SDN. To counteract this threat, the project proposes a predictive security model based on the Random Forest classification algorithm. This model aims to identify malicious behavior by analyzing network traffic patterns. A custom network topology is created using the Mininet tool, while the RYU controller is used for traffic monitoring and for deploying the proposed security solution through a dedicated application. All experiments are conducted in a virtualized test environment to assess the model's accuracy and performance when facing simulated MAC Spoofing attacks under conditions.

Keywords: Mininet, RYU, supervised learning, SDN networks, cybersecurity.

Reviewed by:

Mgs. Sofía Freire Carrillo

ENGLISH PROFESSOR

C.C. 0604257881

CAPÍTULO I

INTRODUCCIÓN

En esta sección, se proporciona información relacionada con las redes SDN y la implementación de herramientas en la seguridad como los modelos de aprendizaje automático, así como los resultados obtenidos de su implementación.

1.1. Antecedentes

En la actualidad, la creación de redes definidas por software (SDN) ha sido posible gracias a los avances tecnológicos, las cuales son diseñadas para que el control del hardware sea separado e implementado en un software denominado controlador. Lo que permite que de una arquitectura centralizada sea programada, automatizada y flexibilizada, al mismo tiempo que se facilita su implementación de forma virtual, independientemente de las redes físicas subyacentes. Estas redes se diferencian de las tradicionales porque las capas de datos y de control son desacopladas, ya que cada una de ellas son automatizadas sin requerir las complicaciones y costos asociados al hardware y software convencionales [1].

Por otra parte, la integración de redes SDN como parte de la infraestructura en el área de las telecomunicaciones como en la tecnología 5G, identificando las limitaciones existentes y las formas de proporcionar seguridad en el paradigma de visualización de redes. Dentro del estudio en el impacto de la seguridad en estas tecnologías, se ha diseñado una solución de gestión de políticas de seguridad sobre el controlador SDN ofreciendo servicios de seguridad. La relación entre los datos recolectados con el modelo de aprendizaje autónomo permite la creación de servicios bajo demanda en la detención de ataques enfocados en la protección de la información de la red [2].

El estudio realizado sobre el manejo de modelos de aprendizaje autónomo (ML) y aprendizaje profundo (DL), como un mecanismo de prevención y detección de amenazas en la infraestructura tradicionales, también se puede enfocar en dar seguridad a las redes SDN. Por tal motivo, darles el enfoque a estos modelos permite el aprendizaje de patrones anormales dentro del tráfico, a diferencia de los mecanismos clásicos, tomando en cuenta el uso de herramientas de código abierto en la creación de prototipos de detención [3].

Al mismo tiempo, el enfoque prometedor en la mejora de la seguridad de las redes IoT mediante la integración de SDN al permitir generar mecanismos de control de acceso porque el Internet de las cosas (sensores/dispositivos) son muy vulnerables a diversos ataques. No obstante, los ataques DDoS representan una amenaza considerable en las redes IoT mediante el uso de computadoras zombis. Por consiguiente, se ha propuesto la detección de estos ataques basándose en modelos de ML, agregando protección al combinar las tecnológicas SND, IoT y ML para abordar los desafíos de seguridad [4].

Cabe considerar que el uso de tecnologías como SDN y herramientas como ML proporcionan diversas oportunidades para mejorar la seguridad en este tipo de redes. También presentan ciertos desafíos, como la falta de estandarización, la variabilidad en los requisitos de costo y recursos necesarios para emplear un modelo ML eficiente. Por un lado, tiene gran relevancia abordar este tipo de retos que permitan desarrollar soluciones de seguridad SDN en conjunto de modelos ML, proporcionando que sea escalable, rentabilidad y eficiencia, la implementación de estos sistemas [5].

Una de las técnicas de ML que se han investigado a mayor profundidad es el aprendizaje por refuerzo (RL) para la seguridad en redes SND. Este tipo de sistemas proporciona la forma de aprender a detectar y mitigar ataques (DDoS), logrando adaptarse a los cambios del flujo del tráfico de la red. Entonces, el uso de herramientas de código abierto, como POX entre otras alternativas, promueven el desarrollo y la implementación de estos sistemas de seguridad basados en RL [6].

1.2. Problema

La progresiva evolución del internet mediante la interconexión de redes convencionales y el incremento de usuarios con fácil acceso a cualquier tipo de medio de comunicación sin considerar las restricciones geográficas y el constante desarrollo de nuevas tecnologías, como el internet de las cosas (IoT), computación en la nube, juegos remotos, Inteligencia artificial (IA). Sin embargo al alta demande de estos servicios y las limitaciones presentes en cuanto al espacio que ocupa una red física [7].

Además, otros de los factores importantes en las redes de datos esta la importancia en la integridad de la información, en el mismo contexto surgen las redes definidas por software SDN como una propuesta centralizada y programable que permita elevar el desempeño de las comunicaciones digitales. Estas características permiten una gestión personalizada más dinámica y eficiente de los recursos. Sin embargo al ser una tecnología emergente su arquitectura permite la introducción de nuevas formas de vectores de ataques debido a la mayor exposición de la capa de control y los requerimientos de gestionar la comunicación de una forma más trasparente [8].

Por lo tanto, el desarrollo de estrategias de seguridad específicas para redes SDN es vital para mitigar los riesgos relacionados con la pérdida o intercepción anormal de los datos. Tomando encuentra, la evolución tecnológica también ha traído consigo nuevos desafíos en términos de seguridad cibernética. Ya que las arquitecturas centralizadas y programables como las presenten en las redes SDN pueden ser susceptibles a una amplia gama de amenazas, como ataques DDoS, infiltraciones de malware y técnicas de manipulación del tráfico de la red y entre otros problemas existentes en este tipo de redes [9].

En este contexto, el desarrollo de estrategias de seguridad en redes SDN mediante procesamiento inteligente de datos surge como una respuesta clave para abordar estas amenazas de manera efectiva. Al aprovechar técnicas avanzadas de análisis de datos, como el aprendizaje automático, la detección de anomalías y la correlación de eventos en el tráfico de la red, es posible fortalecer la seguridad de las redes SDN y mitigar los riesgos asociados con los ataques cibernéticos en este tipo de tecnologías [10].

1.3. Justificación

La necesidad de proteger las infraestructuras de red contra amenazas cibernéticas cada vez es más demandada, ya que estos ataques cada vez son más sofisticados y dinámicos. A continuación, se presentan varios puntos que respaldan esta necesidad.

Aun cuando, las redes SDN sean consideradas como una revolución en cuanto a la gestión y control en las infraestructuras de red al presentar una separación en el plano de control del plano de datos. De igual manera, la composición de esta arquitectura también ha introducido nuevas formas de ataques, lo que ha permitido que las amenazas sean más sofisticadas al detectar con técnicas tradicionales. Las soluciones de seguridad estática no proporcionan las ventajas de la incorporación de machine learning (ML), presentando modelos dinámicos al comportamiento del tráfico e identificar patrones anormales y alertar los ataques [11].

Al considerar, las limitaciones con las técnicas de seguridad convencionales, como Firewalls y sistemas de detección de intrusos (IDS) basados en firmas, porque estos mecanismos requieren constantes actualizaciones y no contemplan la detección de un comportamiento anómalo sin una referencia previa. Los ataques en las redes SDN, pueden lograr comprometer la funcionalidad del controlador, es necesario que las soluciones donde el ML juega un papel esencial, ya que identifica desviaciones estadísticas o patrones inusuales del tráfico, considerando las capacidades predictivas a las respuestas de ataque [12].

La implementación de un modelo de seguridad basado en machine learning que incluye la instalación y configuración correcta de las herramientas necesarias, la implementación correcta de los permisos necesarios en los directorios respectivos a cada programa. Además, la automatización en el análisis y en la gestión del tráfico de la red permitirá prestará nuevos servicios con esta nueva arquitectura [13].

La presente investigación plantea la implementación de un modelo de seguridad en las redes SDN, que permita la alerta de amenazas; también para el entorno seguro de desarrollo y pruebas se emplea en el ambiente de máquinas virtuales dentro de los sistemas basados en UNIX (Linux), con herramientas de código abierto enfocadas para la experimentación, igualmente comprender el funcionamiento de las redes SDN y sus requerimientos [14].

1.4. Objetivos

1.4.1. General

- Implementación de un modelo de seguridad basado en técnicas de machine learning para redes definidas por software (SDN) utilizando herramientas de código abierto.

1.4.2. Específicos

- Investigar sobre las amenazas y vulnerabilidades en redes SDN para comprender su impacto en la seguridad y rendimiento.
- Seleccionar y adaptar algoritmos de machine learning y análisis de comportamiento para la alerta de amenazas.
- Implementar a través de herramientas de código abierto un sistema de alerta basado en datos adaptado a las necesidades de las redes SDN.
- Evaluar la efectividad del sistema propuesto en entorno controlado, para intentar mitigar amenazas y minimizar falsos positivos y negativos.

CAPÍTULO II

MARCO TEÓRICO

En esta sección, se presenta un análisis literario de artículos, libros y textos académicos publicados que se encuentran relacionados con el tema de investigación, abarcando los últimos cinco años.

2.1. Estado del arte

N. Satheesh y su equipo de profesores desarrollaron un modelo basado en prioridades empleando SDN en el control del flujo de los paquetes de la red y también uno de los algoritmos de aprendizaje autónomos basados en Random Forest (RF), Máquina de soporte vectorial (SVM), Árbol de decisiones (DT), para identificar interferencias en la red, con políticas de ancho de banda. Para el entrenamiento y evaluación de los modelos se usó un conjunto de datos NSL-KDD, donde el modelo RF con selección de características por Gain Ratio con una precisión del 82.28%, el modelo SVM obtuvo un 80% y el modelo DT presento la mayor tasa de precisión del 95.16%, en un entorno con las características de 64 GB de memoria y una CPU de 32 núcleos, generando resultados positivos en la detección [15].

El trabajo de los autores J. Bhayo, S. Attique, S. Hameed, A. Ahmed, J. Nasir, D. Draheim, analizan las dificultas que presente los dispositivos que conforma el internet de las cosas (IoT) y presenta un marco de trabajo basado en el aprendizaje autónomo y el uso de las redes SDN combinado con IoT generar redes SD-IoT, con características dinámicas, reprogramables para la detección oportuna de ataques DDoS. Los tres modelos de aprendizaje autónomo seleccionados (Naive Bayes, Árbol de decisión, Máquinas de soporte vectorial), presentaron tiempos de detección semejantes y una tasa de precisión 97.4% (NB), 96.1% (SVM), 98.1% (DT), en cuanto al consumo de recursos el modelo de detección requiere de un 30% de uso de la memoria y la CPU. El enfoque propuesto puede mejorar la seguridad de las redes IoT y mitigar los ataques DDoS [16].

Los autores de la investigación T Anh Tang y su equipo proponen un sistema de detección de instrucciones para una red SDN con un enfoque de aprendizaje profundo (deep learning). Empleando un conjunto de datos NSL-KDD para el entrenamiento y respectiva evaluación de los modelos, logrando una precisión del 80.7% para una red neuronal profunda (DNN) y 90% en una red neuronal recurrente cerrada (GRU-RNN). Lograron confirmar que el enfoque DL presenta el potencial para la detección de anomalías del

flujo de datos en entorno SDN con aspectos positivos en el rendimiento, latencia y utilización de los recursos [17].

El grupo de trabajo de O. Tonkal, describen que las redes SDN por su naturaleza centralizada es propensa a muchos vectores de ataque, como los ataques DDoS y la implementación de algoritmo de aprendizaje autónomo que incorporan análisis de vecinos, en el cual se implementó un conjunto de datos SDN del ataque DDoS que es público el cual consta de 23 características tanto de tráfico normal como del ataque. Estos protocolos son los que se emplearon en la red, como el control de transporte (TCP), datagrama de usuario (UDP) y mensaje de control de internet (ICMP). Además, los algoritmos empleados son k-Vecinos más cercanos (kNN), árbol de decisiones (DT), red neuronal artificial (ANN), máquina de soporte vectorial (SVM), demostraron una precisión 95.58% (kNN), 100% (DT), 97.78% (ANN), 82.04% (SVM). El que mejor resultado presento fue el árbol de decisiones [18].

W. Abdulrhman, H. Hamdi, N. Azim, A. Abd, describen que la integración de IoT en los hogares como sistemas domésticos inteligentes, por sus cualidades en el confort y mejora en la calidad de vida, igualmente la potenciación de estas redes al implementar la visualización y las SDN permite mayor adaptabilidad, administración a un costo menor. El incremento de los dispositivos IoT en conjunto de redes SDN ha generado el interés al valorar el tipo de vulnerabilidades que pueden presentar, como, contra medida, la correcta implementación de medidas de seguridad inteligentes en la detención y mitigación de ataques. El objetivo de esta investigación es el uso de aprendizaje autónomo y profundo como solución, empleando dos conjuntos de datos, el primero de casas inteligentes y el segundo de IoTID20, con una precisión de 99.9% (Xgboost), 98.9% (LSTM) y 85.7% (ANN), manteniendo la operatividad de los dispositivos IoT [19].

2.2. Marco teórico

Para este proyecto, se exploran los componentes de un sistema de seguridad, tomando en cuenta todas las herramientas de software para implementar el prototipo. Asimismo, se considera el impacto que presenta en el entorno controlado por el sistema operativo y la estructura de red SDN, bajo las pruebas en máquinas virtuales.

2.2.1. Hipervisor tipo 2 VirtualBox

Herramienta o software que permite administrar los recursos de un computador (conocido como anfitrión), permitiendo la creación de máquinas virtuales como entornos aislados en los cuales se puede ejecutar su propio sistema operativo. Esto nos permite crear y ejecutar varias máquinas con diferentes sistemas compartiendo los recursos del dispositivo físico, pensado para experimentar, desarrollar y probar diferentes aplicaciones que no están disponibles en el sistema anfitrión [20].

2.2.2. Ubuntu

Una de las distribuciones del ecosistema de Linux derivada de Debian y mantenida por la empresa Canonical, también por la ayuda de una comunidad de voluntarios, todo esto ha permitido que tenga una elevada popularidad tanto en equipos de uso doméstico como en servidores en el ámbito empresarial y académico [21].

2.2.3. Redes SDN

Las tecnologías SDN o redes definidas por software (por sus siglas en inglés Software Defined Networking) introduce un cambio de paradigma en lo que se percibe en cuanto al desarrollo y despliegue en las redes convencionales de datos. Su funcionamiento está basado en la separación del plano de datos (infraestructura que reenvía los paquetes) [22], el plano de control (manera en cómo se reenvían los paquetes), como se muestra en la red de topología mesh de la figura 1. Ventajas que presenta este tipo de redes:

- **Simplificación en la gestión:** Al estar el plano de control separado, facilita la configuración automatizada de políticas, servicios o aplicaciones de la red al permitir modificaciones y optimizaciones [13].
- **Tiene escalabilidad:** De acuerdo con las necesidades del servicio, la infraestructura de la red, al ser programable, acarrea un menor tiempo de ejecución, ya que los cambios que se le realicen son mediante API’s sobre el controlador, modificando el comportamiento de los conmutadores [19].

- **Reducción de latencia:** Considerando que el plano de control es centralizado, los conmutadores trabajan con las instrucciones del controlador brindadas a través de la tabla de flujo, esto permite que cada paquete entrante se empareje con el puerto de salida del conmutador y las instrucciones de destino, siendo más eficiente que las redes convencionales donde pasa por toda la infraestructura de la red asta poder identificar su destino [15].
- **Herramienta de código abierto:** Considerando instituciones como la Linux Foundation o la ONF, el desarrollo de las redes SDN es de carácter de código abierto; esto garantiza la accesibilidad, libertad y gratitud en los posteriores trabajos o investigaciones que empleen esta tecnología [13].

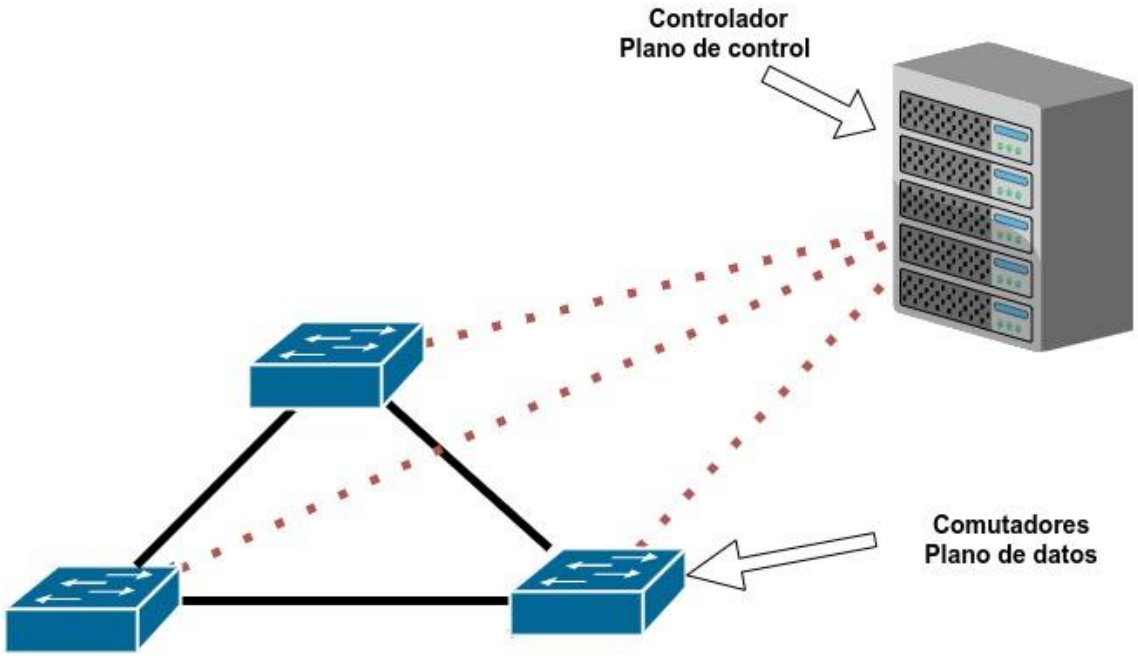


Figura 1. Estructura básica de una red definida por software.

2.2.4. Arquitectura de redes SDN

Este tipo de redes proporcionan mayor flexibilidad y gestión al ser altamente programable, su estructura principal consta de niveles: capa aplicación, capa de control, capa de infraestructura [23], la comunicación entre cada una de estas capas se hace mediante interfases de programación (API), de acuerdo con la figura 2:

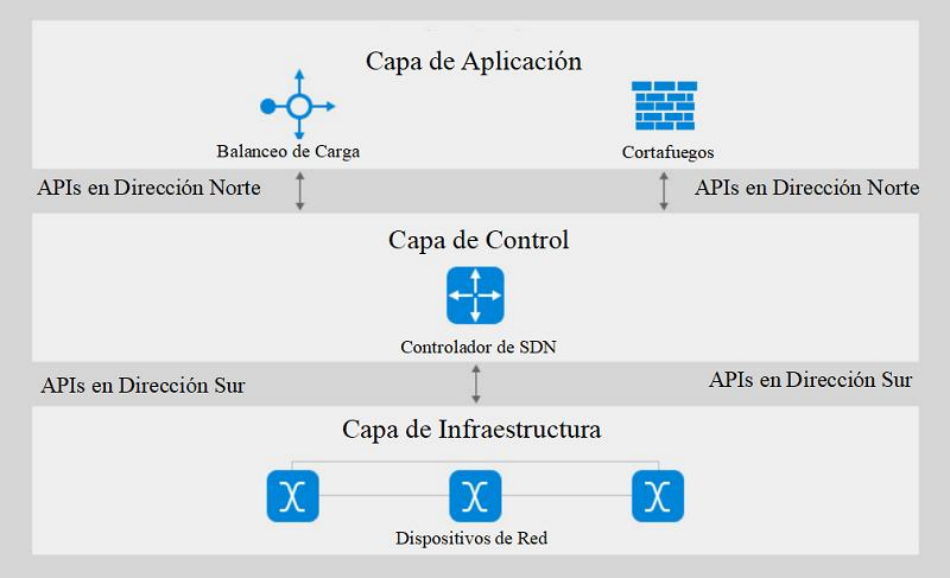


Figura 2. Arquitectura de redes SDN.

- **Capa de aplicación:** En este apartado se tienen las aplicaciones y servicios que interactúan con la infraestructura de la red a través de las API con el controlador. Las funciones de estas aplicaciones van desde la gestión de tráfico, seguridad, monitoreo de redes y optimización de recurso [24].
- **Capa de control:** El controlador considerado como el corazón de las redes SDN, al ser un componente centralizado en el cual se plantean las decisiones del manejo del tráfico de la red en relación con las políticas definidas por las aplicaciones y servicios [25]. La comunicación del controlador se da por medio del protocolo OpenFlow al programar y gestionar el comportamiento de los switches y routers.
- **Capa de Infraestructura:** Se comprende como los dispositivos de red física (switches, routers), estos dispositivos conocidos como elementos reenviados (forwarding elements), se encargan de mover los paquetes de datos de un punto a otro de la red acorde con las instrucciones del controlador [25]
- **Interfases de Programación (API):** Estos protocolos permiten la comunicación de las capas inferiores hacia las superiores, al igual que el controlador, permitiendo a las aplicaciones enviar solicitudes al controlador para programar,

gestionar el comportamiento de la red y recibir información sobre el estado de esta. API en dirección sur permite la comunicación del plano de datos y el de control, y la API en dirección norte genera la comunicación entre el plano de aplicación y el de control [26].

2.2.5. Protocolo OpenFlow

Desarrollado en el 2008, por un grupo de investigadores de la Universidad de Stanford, liderado por Nick McKeown, considerado como forma innovadora de explorar las redes existentes, aunque al principio no se definía de forma muy clara la relación entre redes SDN y OpenFlow. Con la creación de un controlador de software independiente de la infraestructura de este tipo de redes, generando la estandarización de la comunicación entre el controlador y los dispositivos de red, al permitir la programación de tablas de flujo en el plano de aplicación y estableciendo los procedimientos para trasferir el control de la red por el plano de control [27]. Un dispositivo con OpenFlow constan de al menos tres componentes principales:

- **Tabla de flujo:** Esta tabla contiene las entradas de los campos que se utilizan para buscar conciencias con los paquetes entrantes, controladores para estadísticas de paquetes y proporcionar instrucciones sobre el manejo en las conciencias de los paquetes [24].
- **Canal seguro:** Canal que conecta los dispositivos con el controlador, permitiendo la comunicación bidireccional de comandos y paquetes por medio del protocolo OpenFlow [28].
- **Protocolo OpenFlow:** Es un estándar abierto para la comunicación entre los conmutadores y el controlador, logrando que el controlador inserte, elimine, modifique y busque entradas en la tabla de flujo a través del canal seguro [23].

2.2.6. Controlador SDN

Se puede entender como el cerebro lógico de las redes SDN, al proporcionar una visión más centralizada de toda la infraestructura de la red. De forma concreta, el controlador es una aplicación de software o un conjunto de aplicaciones que permiten la comunicación con los dispositivos de la red (switches, routers, hosts) al emplear el protocolo OpenFlow, también incorpora APIs para que las aplicaciones de red y los sistemas de gestión tengan una iteración con la red de forma abstracta y programable [28]. Características principales de los controladores:

- **Gestión en la topología de red:** El controlador analiza y actualiza constantemente su visión de la topología de red, considerando los dispositivos conectados, la capacidad de estos y los enlaces entre ellos [25].
- **Programación de los dispositivos de red:** Al emplear el protocolo OpenFlow en la configuración de tablas de flujo entre los switches y routers, al determinar cómo se reenvía el tráfico.
- **Recopilación de la información de la red:** Se puede promocionar la recopilación estadística de eventos entre otra información relevante en la disponibilidad de la red para monitorizar, analizar y tomar decisiones [26].
- **Implementación de políticas de red:** Entiende las políticas de red definidas por los administradores en reglas específicas que se gestionan en los dispositivos de reenvíos.
- **Automatización de tareas en la red:** La automatización de actividades por el aprovisionamiento de nuevos servicios en la configuración de dispositivos y la respuesta en eventos de la red [24]
- **Conceptualización de la infraestructura:** Al presentar una visión lógica e ideal de la red a las aplicaciones, ya que los detalles de la infraestructura física no son muy considerados.
- **Control del flujo del tráfico:** Presenta decisiones en el enrutamiento y la gestión en el tráfico de acuerdo con las políticas y condiciones de la red [25].

En el mercado actual existen distintos tipos de controladores SDN, los cuales pueden ser de código abierto o proporcionados de proveedores especializados en el área de redes. Muchos de estos presentan variaciones en cómo se construyen las arquitecturas y se especifican las características. Por el contrario, todos aportan con las funciones necesarias, como la toma de decisiones en el proceso de enrutamiento del tráfico y como punto central para la programación de la arquitectura de la red SDN [29]. Los controladores más conocidos, por ser de ámbito público y de manera gratuita, son:

- **OpenDaylight (ODL):** Uno de los controladores más populares que es de código abierto, el cual está mantenido por la Fundación Linux. Esta herramienta proporciona una plataforma de desarrollo modular y extensible con la posibilidad de añadir funciones y protocolos acordes a las necesidades que se pretenda cubrir [30].
- **Floodlight:** Desarrollado en el lenguaje de programación Java, un controlador muy popular debido a su facilidad de uso y una interfaz de programación API bien diseñada. Este software funcionará con cualquier dispositivo de red compatible con el protocolo OpenFlow; también hay que considerar que es de código abierto. [31]
- **ONOS (Open Network Operating System):** Controlador de código abierto que también cuenta con buena fama, el cual fue desarrollado para cubrir los requerimientos de los operadores de redes. Además, presenta un buen rendimiento, escalabilidad, disponibilidad y abstracción en el desarrollo de aplicaciones y servicios en las redes [32].
- **Ryu:** Este es un controlador de código abierto desarrollado en el lenguaje de Python, generando un enfoque modular y flexible, permitiendo ser otra opción en el área de la investigación académica y en pruebas experimentales sobre redes SDN.
- **HP VAN (Virtual Application Networks) SDN Controller:** Solución comercial desarrollada por la empresa HP que proporciona características de SDN y forma parte de la suite de aplicaciones de la red de HP [26].
- **Cisco APIC-EM:** Proporcionado por Cisco, este controlador SDN forma parte de la arquitectura de red empresarial, centrándose en la simplificación de las operaciones en entornos de campus, sucursales y WAN [26].
- **Juniper Contrail:** Plataforma de virtualización de redes y automatización de servicios, proporcionada por la empresa Juniper Networks de código abierto y basado en estándares. Presenta las características de gestión de redes virtuales en entornos de la nube, aprovisionamiento de autoservicios [25].

2.2.7. Controlador RYU

Los orígenes de este framework escritorio en Python se dieron en Japón 2012 por parte de los laboratorios Software Innovation Center (SIC) y Nippon Telegraph and Telephone Corporation (NTT), bajo la licencia de código abierto Apache 2.0. Logrando que sea de código abierto al integrar varios componentes útiles para el desarrollo de aplicaciones de control para redes SDN, soportando el protocolo OpenFlow en sus versiones 1.0, 1.2, 1.3, 1.4, 1.5 al ser de diseño modular y basado en eventos permitiendo que sea flexible, fácil de entender y expandir [24].

El significado de la palabra 'RYU' en la cultura japonesa hace referencia a los tradicionales dragos orientales, con su estilo está representado en la figura 3. Por otra parte el apartado de la arquitectura se presenta los siguientes conceptos:

- **RyuApp:** Es la forma en la que se interpreta que la aplicación corresponde al controlador Ryu; por otra parte, se entiende que estas aplicaciones pueden ser desarrolladas para necesidades específicas [33].
- **Eventos:** Es la forma de gestionar las alertas presentadas por el propio framework Ryu o switches OpenFlow y las RyuApp registran y procesan eventos únicos [34].
- **Handlers de Eventos:** Son métodos que forman parte de las RyuApp que se emplean al recibir eventos registrados, ya que estos handles son de lógica de control [35].
- **Objetivos de API:** Estas intrusiones son importantes en el desarrollo de las RyuApps porque permiten la interacción con la red mediante de: Datapath (conexión con switch OpenFlow), OFPMessage (mensajes del protocolo OpenFlow), OFPAction (acciones que el switch realiza), OFPMatch (Campos de coincidencia en las entradas de flujo), Topology (información sobre la topología de la red) [36].
- **Services:** Ryu proporciona varios servicios que las aplicaciones los pueden emplear, tales como el descubrimiento de topologías y el aprendizaje de MAC [37].

Figura 3. Logo característico del controlador RYU.

2.2.8. Mininet

Es una herramienta enfocada en el plano de datos, programada en el lenguaje Python, trabaja en el plano de datos proporcionando virtualización, brindado una colección de hosts finales, conmutadores, enlaces y controladores en un único núcleo (kernel) de Linux, permitiendo realizar experimentos en un entono controlado al aislar sus componentes. Presenta una interfaz de líneas de comandos (CLI) en la cual los usuarios pueden interactuar y verificar las configuraciones de la red simulada, también contiene una herramienta llamada MiniEdit que nos permite de forma más didáctica la creación de redes personalizadas como se muestra en la figura 4 con una topología en árbol y exportar lo a un archivo script de Python para ser ejecutado en Mininet y comprobar la comunicación en la red empelando comandos como ping o tcpdump [25]

Además, Mininet es un proyecto de código abierto distribuye bajo la licencia BSD permisiva, lo que indica que cualquier usuario puede contribuir a en su desarrollo, corrigiendo fallos y mantenerlo constantemente actualizado la herramienta y que sea accesible para toda la comunidad de usuarios interesados en la experimentación con SDN y OpenFlow. Para la interacción con los hosts de Mininet el usuario puede acceder mediante el programa Xterm como terminal de estos dispositivos [24].

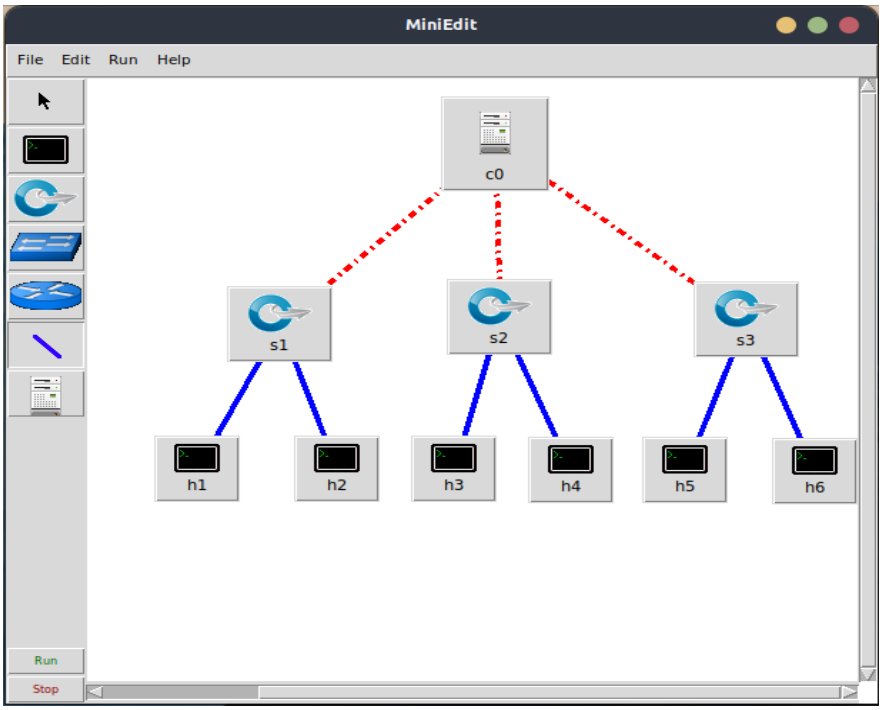


Figura 4. Interfaz de MiniEdit y un pequeño esquema de red

2.2.9. Herramienta IPERF

Una utilidad muy empleada en la medición y análisis del rendimiento al generar flujo en determinados puntos de la red, permitiendo saber la velocidad de trasferencia, capacidad del ancho de banda y otros parámetros claves. La modalidad de funcionamiento es la trasmisión de datos TCP (protocolo de control de trasmisión) y UDP (Protocola de datagrama de usuario) por la red [38].

2.2.10. Datos de entrenamiento (Dataset)

Una utilidad muy empleada en la medición y análisis del rendimiento al generar flujo en determinados puntos de la red, permitiendo saber la velocidad de trasferencia, capacidad del ancho de banda y otros parámetros claves. La modalidad de funcionamiento es la trasmisión de datos TCP (protocolo de control de trasmisión) y UDP (Protocola de datagrama de usuario) por la red [26].

2.2.11. Machine Learning

Considerado como uno de los campos de la rama tecnológica de la Inteligencia Artificial, que logren hacer que los sistemas informáticos aprendan sin programar mediante datos que se les proporcionan y mediante eso puedan realizar una actividad. Otra forma de comprender su funcionamiento es que no requiere de instrucciones, sino que mediante el uso de algoritmos logra identificar patrones, extracción de información, tomar decisiones y predicciones por medio de los datos [39].

2.2.12. Tipos de machine learning supervisado

Este tipo de modelos tiene como objetivo a partir de datos etiquetados para que se pueda predecir la salida de una forma correcta para datos nuevos y no vistos [40]. Los algoritmos más empleados son:

- **Clasificación:** es la asignación de una instancia de datos a una de varias categorías predefinidas, al distinguir entre diferentes clases basándose en las características de las entradas [41].
- **Regresión:** Tiene como objetivo predecir un valor numérico continuo para un grupo determinado de datos [41].
- **Segmentación:** En este caso, el conjunto de datos está divididos en grupos o segmentos, para el caso de que los elementos de una misma categoría son similares a las relaciones [42].
- **Clasificación Multietiquetas:** Cuando se trata de clasificación multietiqueta, en este caso de que una instancia pueda pertenecer a más de una categoría [42].
- **Clasificación Multiclases:** Presenta similitudes con cuanto lo hace la clasificación, pero para este caso la instancia se clasifica en una de varias categorías posibles [43].
- **Detención de anomalías:** En estos casos, el objetivo es identificar las instancias que son inusuales o anormales en relación con el resto de datos [43].

2.2.13. Random Forest

Uno de los algoritmos de aprendizaje supervisado de conjuntos más conocidos y utilizados, al presentar un modelo que combina modelos autónomos simples, lo cual permite predicciones finales y más robustas. Estos modelos simples son agrupaciones de árboles de decisiones, lo que le da sentido al nombre 'Forest' o bosque, presenta las ventajas de que es un algoritmo que permite los modos de clasificación, regresión y presenta buenos resultados de predicción [32], [44].

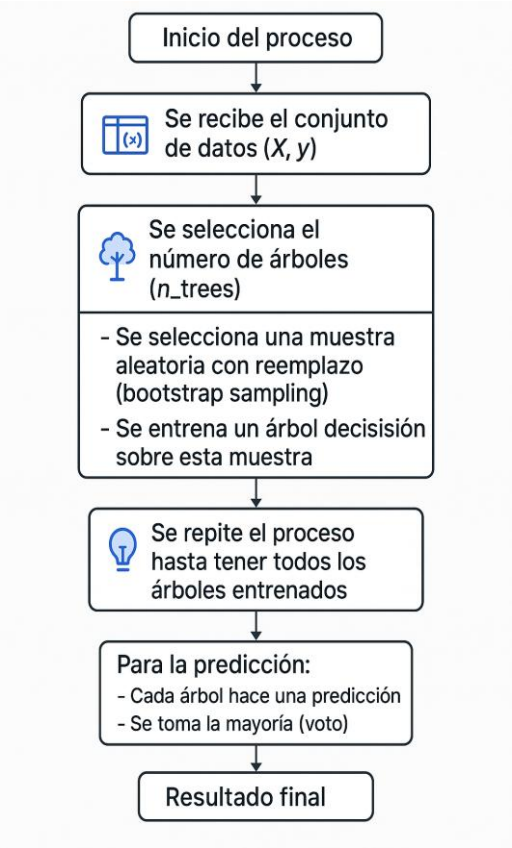


Figura 5. Diagrama de flujo del algoritmo Random Forest

Para la figura 5 se da a conocer el diagrama de flujo para este algoritmo y su parte matemática está contenida dentro de la tabla 1.

Tabla 1. Interpretación matemático algoritmo (random forest).

Fórmula	Descripción
$\mathrm{D} = \{(\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \dots, (\mathrm{x}_\mathrm{N}, \mathrm{y}_\mathrm{N})\}$	Interpretación del conjunto de entrenamiento, donde $x_i \epsilon R^d$ son vectores de características y $\mathbf{y_i} \epsilon \{\mathbf{1}, \mathbf{2}, \dots, \mathbf{C}\}$ son las etiquetas de clases
En la generación de bosque aleatorio, cada árbol de decisiones, $\mathbf{T_j}$, se genera una muestra de entrenamiento $\mathbf{D_j}$ a través de muestro con reemplazo y en cada nodo del árbol, los subconjuntos ($\mathbf{m} \ll \mathbf{d}$) son seleccionados de forma aleatorio, para encontrar la mejor división.	
$\mathrm{H}(\mathrm{x}) = \mathrm{mode}(\{\mathrm{h}_1(\mathrm{x}), \mathrm{h}_2(\mathrm{x}), \dots, \mathrm{h}_\mathrm{K}(\mathrm{x})\})$	La predicción final, para una entrada x la cual mayor número de votos entre todos los árboles presento; $\boldsymbol{K}$es el numero total de árboles, $mode$ es la moda o la clase más votada.

2.2.14. Redes SDN y su seguridad

Uno de los problemas más comunes en las redes tradicionales y en general las nuevas tecnologías es el avance y la sofisticación de los vectores de ataque. Al considerar sobre las soluciones de seguridad convencionales que no evolucionan con el paso de tiempo, pierden su efectividad ante las mejoras que se dan en la forma que actúan los ataques. Otro apartado a destacar son las vulnerabilidades de los equipos y herramientas de software, ya que estos fallos son aprovechados por los ciberdelincuentes comprometiendo las infraestructuras de entidades o empresas con fines económicos [44].

No obstante, recordar las características de las redes SDN, como su arquitectura y la amplia flexibilidad y programabilidad en el desarrollo de este tipo de redes, no obstante, presenta desafíos en relación con nuevas formas de ataque [45]. Por último se presenta en la tabla 2 los tipos de ataques más relevantes en redes SDN.

Tabla 2. Vulnerabilidades y Amenazas redes SDN.

Controlador SDN		Ataques al controlador	Compromete el controlador.
		Vulnerabilidades	Explotación de vulnerabilidades en el controlador.
		Denegación de servicios (DDoS)	Saturación e inaccesibilidad al controlador.
Canal de comunicaciones	**Interfaz Southbound**	Interceptación y manipulación	Captura y manipulación de los mensajes de control.
		Ataque 'Man-in-the-Middle'	Obtener comunicación entre el controlador y los switches.
	Interfaz Northbound	Vulnerabilidad de API	Explotación de vulnerabilidades en las API.
		Acceso no autorizado	API no protegidas; manipulación del controlador.
Aplicaciones SDN		Aplicaciones maliciosas o comprometidas	Vulnerabilidades o programa maliciosos.
		Errores de configuración	Brechas de seguridad por error humano
Plano de datos		Manipulación de reglas de flujo	Comprometer el controlador o el canal de comunicaciones
		Ataque de inundación (flooding)	Inundación de MAC o a las tablas de flujo
		Spoofing	Ataques falsificación de MAC o IP

2.2.15. Ataque Spoofing MAC

Técnica en la cual el atacante suplanta de un dispositivo legitimo con la finalidad de poder interceptar todo el tráfico o que contantemente se encuentre cambiando de dirección MAC mientras establece comunicación con algún host autentico con la finalidad de no ser detectado y obtener el tráfico de la red, generando una sobrecarga en la infraestructura de la red [31].

Figura 6. Funcionamiento del ataque SPOOFING MAC.

Para el ejemplo de la figura 6 tenemos una maquina atacante que está enviando de forma continua paquetes de datos generados con direcciones MAC aleatorias a la víctima.

2.2.16. Modelo de seguridad implementado machine learning

Técnica de seguridad que implementa un conjunto de herramientas (software) que permitan cumplir una o varias acciones de protección, en el caso de las redes definidas por software (SDN), el enfoque en alguna área de la infraestructura, como la protección de sus datos, accesos no autorizados e interrupciones. Para este modelo una de las herramientas que se tomarán en cuenta son los algoritmos de machine learning [45].

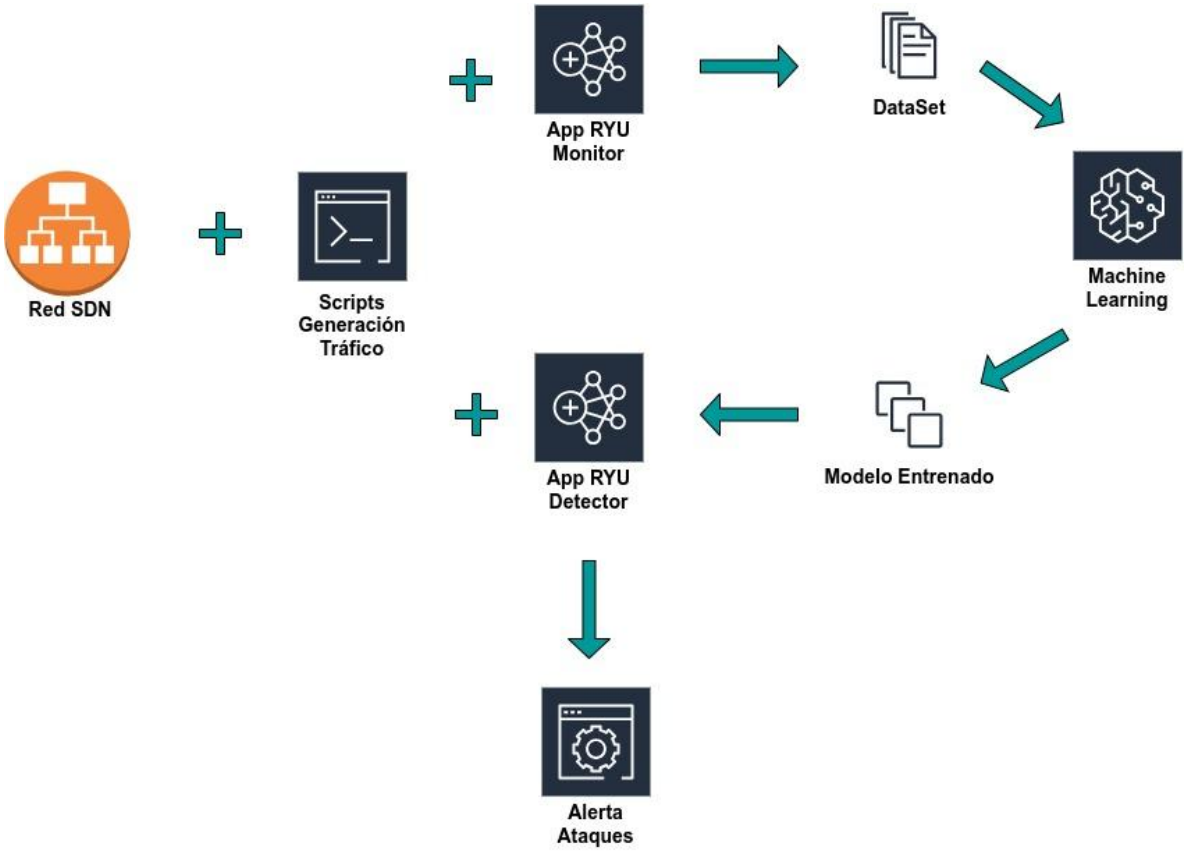


Figura 7. Modelo de seguridad empleando machine learning para redes SDN

La estructura del prototipo de modelo de seguridad (figura 7), para este proyecto en particular presenta consta del despliegue de la red, la ejecución del tráfico y su respectiva captura de datos en archivos DataSet, entrenamiento de los modelos con estos archivos e implementación en el sistema para generar las pruebas de ataque.

CAPÍTULO III

METODOLOGÍA

En esta sección, se lleva a cabo la investigación inicial sobre el desarrollo de red definidas por software y las herramientas de código abierto, modelos de aprendizaje autónomo disponibles, A continuación, se abordó el modelo de seguridad, definiendo los requisitos específicos y seleccionado los componentes más adecuados. Durante la fase de desarrollo, se creó un prototipo del modelo de seguridad funcional. Más adelante, se realizaron evaluaciones para verificar los tiempos en la detección en un entorno controlado.

3.1. Tipo de investigación

Este proyecto de investigación se considera de carácter experiencial, mediante el cual se evaluó la seguridad del modelo, en la identificación del ataque con relación a al número de vectores maliciosos generados en la red SDN. También se usó un enfoque cuantitativo para recopilar y analizar el tráfico normal y bajo ataque de la red en Mininet [46].

3.1.1. Técnica de documentación

La recopilación de información se obtuvo de fuentes como depósitos digitales de investigación, libros, trabajos de investigación y tesis asociados con el tema de estudio, estos archivos se agruparon en dos niveles de importancia, para el primer caso reconocer las amenazas en redes SDN, su impacto en la seguridad y el segundo caso identificar el uso de machine learning para la alerta de amenazas con herramientas de código abierto [30].

3.2. Técnicas de recolección de datos

Se llevó a cabo mediante la observación directa y el registro de los eventos en el caso de las alertas de ataques o vector de ataque presentado por el sistema y un script que permita la simulación correcta y genere el tráfico malicioso para una red SDN en un entorno controlado como parte del proceso de experimentación [47].

3.3. Población de estudio y tamaño de muestra

La población obtenida en este estudio comprende los datos considerados como vectores de ataque que son generados en la arquitectura de las redes SDN, todo esto se genera dentro de un entorno de desarrollo seguro y aislado, empleando con el algoritmo se genera modelos de entrenamiento que permitan al sistema detectar de forma dinámica los ataques [48]. Al emplear un entorno virtual, no se cuenta definido el número de ataques que puede recibir una red SDN de forma real, empleando la fórmula de población infinita (1), expresada de la siguiente forma:

$$n = \frac{Z^2 * P * Q}{e^2} \tag{1}$$

$$n = \frac{(1.86)^2 * (0.50) * (0.50)}{(0.05)^2} \cong 384$$

Donde:
n = Tamaño de la muestra
Z = Nivel de confianza de 95%, con su puntuación estándar de 1.96
P = Probabilidad que ocurra en el caso de estudio de 50%
Q = Probabilidad que no ocurra en el caso de estudio 50%
e = Error de estimación, que es 4% = 0.05

3.3.1. Operacionalización de variable

Tabla 3. Variable dependiente.

Variables	Descripción	Método	Instrumento	Indicador
Ataques identificados	Cantidad de tráfico malicioso reconocido por el módulo de seguridad	Observación	Log o archivos de registro, alerta por consola	Porcentaje de ataques registrados

Tabla 4. Variable independiente.

Variables	Descripción	Método	Instrumento	Indicador
Número de ataques	Ataques MAC Spoofing generados en la red SDN	Experimentación	Script Python	Tráfico generado

3.4. Hipótesis

H0: La distribución de precisión en el número de ataques detectados es la mismas en los 3 modelos.

Ha: La distribución de precisión en el número de ataques es diferente al menos en uno de los 3 modelos.

3.5. Métodos de análisis y procesamiento de datos

En la comprensión del análisis y procesamiento de datos sobre una muestra de 384 se trabajó con 300 ataques de para validar el funcionamiento de los modelos ya entrenados mediante machine learning y que estos son herramienta claves en las pruebas realizadas al sistema de seguridad de alertas dentro del entorno controlado [49].

3.6. Proceso de la metodología

En este apartado se aborda cada uno de los pasos en el desarrollo de la metodología de modelo de seguridad basado en machine learning.

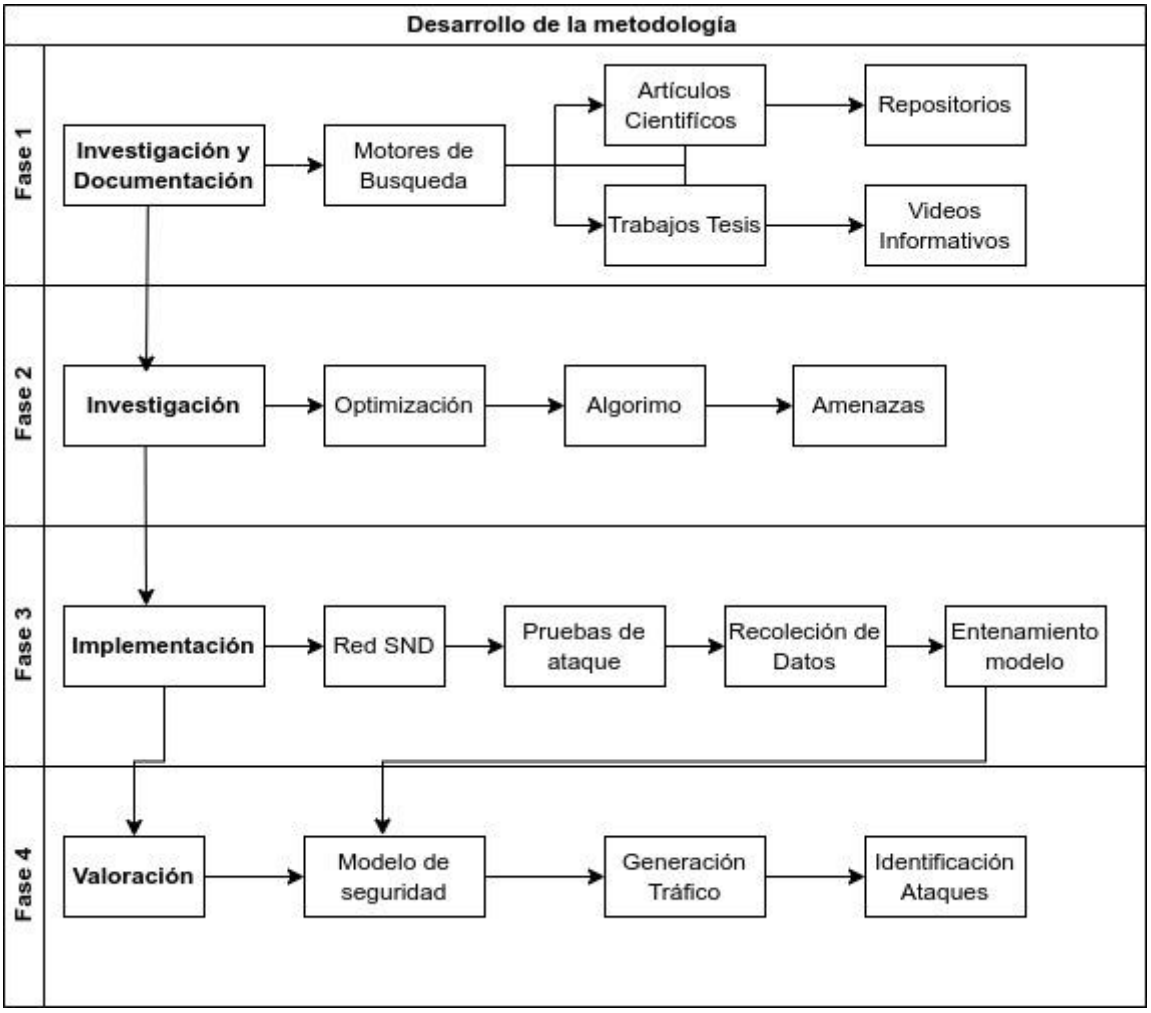


Figura 8. Fases del desarrollo del proyecto.

Mediante la figura 8 se tiene una visualización de forma general sobre las fases del desarrollo del proyecto, estas fases se expresan más adelante.

Fase 1: Investigación y documentación

La indagación sobre documentación relaciona con temas, como ‘redes definidas por software’, ‘manejo del controlador RYU’, ‘algoritmos de machine learning’, ‘amenazas y seguridad presente en redes’, ‘herramientas de código abierto’, para lo cual se llevó a cabo por medio de motores de búsqueda al acceder a repositorios digitales. Clasificando cada uno de los documentos en relación con los temas de interés y palabras claves enfocadas con el proyecto. Los sitios donde se encuentran las bases de datos académicas, más extensas, son: IEE explore, Google Scholar, El Sevier, MPDI, Mendelay, entre otros. En esta fase se recopiló análisis y organización de la documentación, obteniendo 10 artículos científicos, 16 tesis, 4 libros, como parte de desarrollo del proyecto y fuentes para el mismo.

Fase 2: Selección y adaptación de algoritmo de machine learning

En esta parte de la investigación se analizaron de forma relevante a los tipos de algoritmos existentes, características y cualidades mediante los análisis estadísticos de investigaciones previas. Una vez establecido el tipo de algoritmo óptimo dentro del campo de las amenazas en redes de la misma arquitectura y de las clásicas redes. Dentro de este proceso también se estableció el lenguaje de programación a emplear ‘Python’ en el desarrollo de barias áreas del sistema de seguridad.

Fase 3: Implementación del modelo de seguridad

En este apartado se incorporaron todos los conocimientos obtenidos de las etapas anteriores, en especial de videos informativos y foros relacionados con el tema, permitiendo ser una base para el desarrollo del modelo de seguridad. La materialización del proyecto comenzó con la selección de las condiciones necesarias para el modelo de seguridad, con la instalación de librerías y programas necesarios como, sklearn, scapy, pandas, Mininet, RYU.

Después de establecer los requerimientos necesarios, se genera la estructura personalizada en Mininet, el script en el cual se puede controlar el número de paquetes a enviar, el intervalo de tiempo y la dirección MAC de la víctima para la generación de tráfico malicioso, Configuración del controlador RYU como monitor de la red, obtención de registros de todo el tráfico de la red, entrenamiento del algoritmo de machine learning, aplicar el modelo entrenado, aplicación de detección empleando el modelo entrenado.

Fase 4: Evaluación del modelo de seguridad en un entorno controlado

En esta parte final del proyecto, se generaron las pruebas de alerta del tráfico malicioso dentro del entorno controlado y verificaron la incorporación del machine learning entrenado.

El proceso de evaluación comienza con cuando el modelo entrenado con los datos previos del tráfico es incorporado a la aplicación de detección en RYU, el despliegue de la red en Mininet y el controlador RYU con la respectiva aplicación, para posteriormente en cada uno de los hosts de la red simular el tráfico de la red y los ataques.

3.7. Características de desarrollo del modelo de seguridad

Los requerimientos, bibliotecas y herramientas necesarias para el desarrollo del prototipo, comienzan con el levantamiento del entorno seguro en una máquina virtual para poder proporcionar las pruebas correspondientes sin comprometer cualquier tipo de infraestructura o dispositivo.

3.7.1. Creación del entorno de pruebas (Máquina Virtual)

Una de las alternativas de hipervisor que se consideró es VirtualBox, como la base del desarrollo el ecosistema controlado, como Sistema Operativo adecuado para emplear las herramientas de código abierto, considerando las características del equipo anfitrión tabla 5, se le asignó los recursos necesarios a la máquina virtual tabla 6 [48].

Tabla 5. Características del equipo anfitrión.

Marca	HP
Sistema	Windows 11
Almacenamiento	500 GB
Memoria RAM	16 GB
Procesador	Intel Core i5 7th Generación
Numero de núcleos lógicos CPU	4
Configuración de red	WIFI – Ethernet

Tabla 6. Características de la máquina virtual.

Hipervisor	VirtualBox
Sistema	Ubuntu 20.04.6 LTS
Almacenamiento	25 GB
Memoria RAM	4 GB
Procesador	Intel Core i5 7th Generación
Numero de núcleos lógicos CPU	2
Configuración de red	NAT/Adaptador Puente

3.7.2. Infraestructura de red en Mininet

La estructura de la red se planificó de forma personalizada fue diseñada dentro del programa Mininet, en la cual nos permita desplegar este tipo de redes SDN, presenta varias formas de instalación de esta herramienta para el desarrollo del proyecto se instaló y configuro dentro de la máquina virtual en Ubuntu [38].
La topología personalizada se presenta en la figura 9.

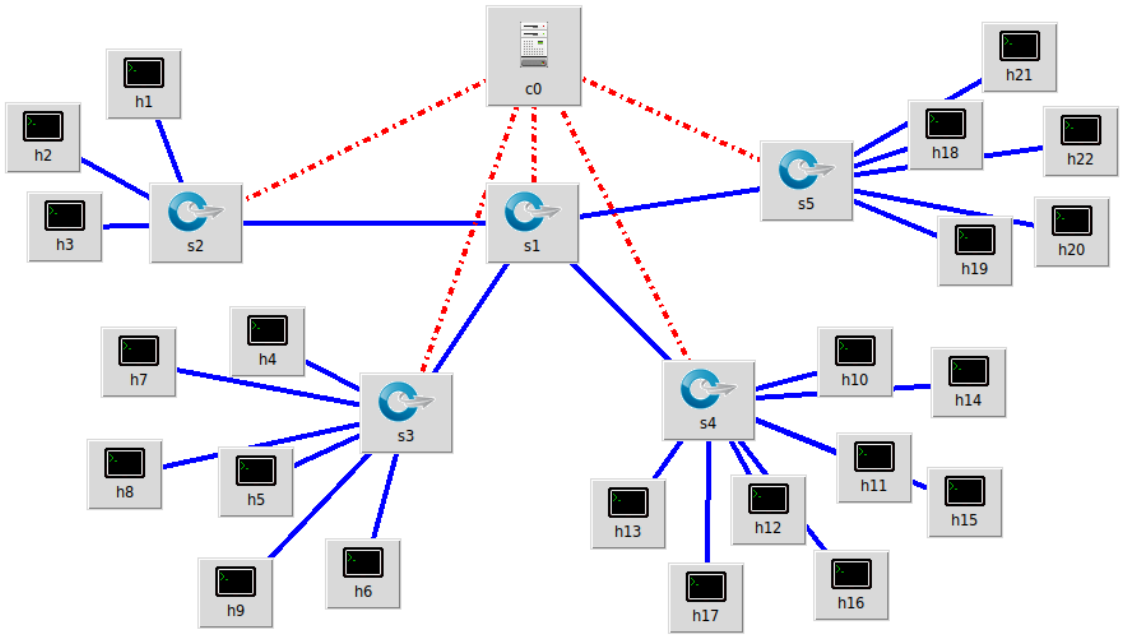


Figura 9. Topología de la red de pruebas.

Para el desarrollo de esta topología (figura 9) es implementada mediante el código en Python, la cual se estructura en 5 switches y 22 hosts. El switch **S1** está conectado de forma directa entre el controlador **C0** y también con los otros switches **S2** al **S5**: para el

segundo switch incorpora 3 hosts, el tercero tiene 6 hosts, cuarto tiene 8 hosts, quinto tiene 5 hosts.
La dirección IP de la red es 192.168.50.0 /24 y a partir de la dirección 192.168.50.2 hasta la 192.168.50.23 y como Gateway la 192.168.50.1, también se personalizó la asignación para cada una de las direcciones MAC de los hosts.

```
info("*** Añadir switches\n")
    s1 = net.addSwitch("s1", cls=OVSSwitch,protocols='OpenFlow13', dpid='00001A2FBBCCDD00')
    s2 = net.addSwitch("s2", cls=OVSSwitch,protocols='OpenFlow13', dpid='00000E7FAABBCC00')
    s3 = net.addSwitch("s3", cls=OVSSwitch,protocols='OpenFlow13', dpid='000005855DEEFF00')
    s4 = net.addSwitch("s4", cls=OVSSwitch,protocols='OpenFlow13', dpid='00001E4FCCAABB00')
    s5 = net.addSwitch("s5", cls=OVSSwitch,protocols='OpenFlow13', dpid='0000049600DDCC00')
```

Script 1. Código de la asignación de los switches.

Esta parte del código en el Script 1, forma parte del código de la red personalizada; también en este punto se establece el protocolo ‘OpenFlow13’para cada uno de los 5 switches.

```
info("*** Añadir hosts\n")
    # Hosts conectados a switch 2
    h1 = net.addHost("h1", ip="192.168.50.2/24", defaultRoute="via 192.168.50.1",
mac="00:25:B3:64:8E:25")
    h2 = net.addHost("h2", ip="192.168.50.3/24", defaultRoute="via 192.168.50.1",
mac="00:21:5A:B7:3F:E8")
    h3 = net.addHost("h3", ip="192.168.50.4/24", defaultRoute="via 192.168.50.1",
mac="00:17:EE:48:C2:A6")
```

Script 2. Código de la creación de los hosts.

Para esta parte del Script 2, sé asigna y configura las direcciones IP y MAC a cada uno de los hosts, además esta parte del código se repite para cada uno de los 22 hosts.

```
info("*** Creación de enlaces\n")
    # Conexión con de los switches con el switch central - 1Gbps con 0.5ms de delay
    net.addLink(s1, s2, cls=TCLink, bw=1000, delay='0.5ms')
    net.addLink(s1, s3, cls=TCLink, bw=1000, delay='0.5ms')
    net.addLink(s1, s4, cls=TCLink, bw=1000, delay='0.5ms')
    net.addLink(s1, s5, cls=TCLink, bw=1000, delay='0.5ms')
```

Script 3. Código de la conexión entre los switches.

Para el código del Script 3, se especifica las características de conexión entre los switches del 2 al 5 con el switch 1 central, estos enlaces tienen una velocidad de 1 Gbps, de forma similar se presenta la conexión entre los switches y los hosts del Script 4, configurado con una velocidad de 100 Mbps. Las configuraciones de las velocidades se hacen con la finalidad de que sea más parecido a un real empresarial.

```
# Conexión de los hosts al switch 3 - 100Mbps con 1ms de delay
    net.addLink(h4, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h5, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h6, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h7, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h8, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h9, s3, cls=TCLink, bw=100, delay='1ms')
```

Script 4. Código de la conexión entre los switches y los hosts.

Para ejecutar la topología de red es necesario ya tener primeramente configurado el controlador RYU, entonces de esta forma se despliega la red correctamente.

```
❯ sudo python3 A1_red_custo9.py
```

Script 5. Código de ejecución de la topología.

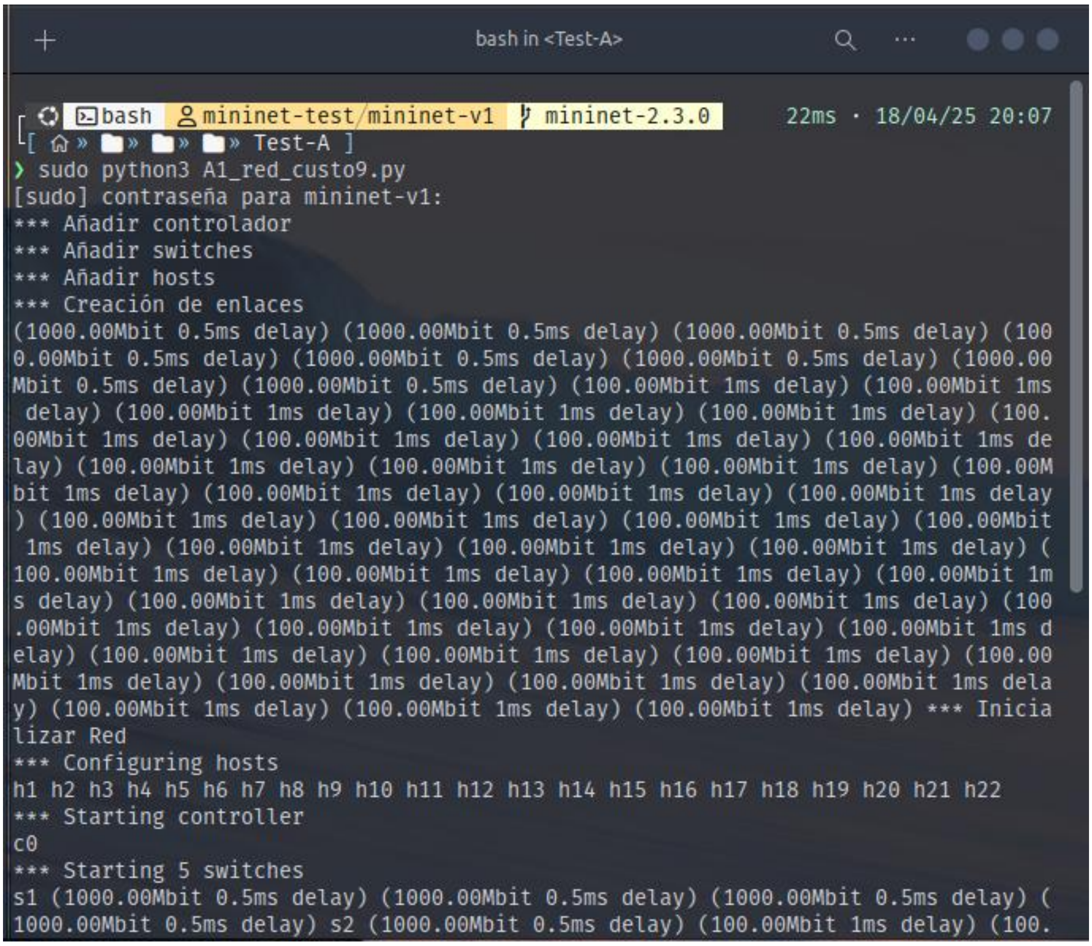


Figura 10. Inicialización de la topología de red.

En la figura 10 se puede apreciar el despliegue de la topología de red personalizada la estructura completa es en anexo 1, con la cual se puede interactuar mediante la consola de comandos [50]. Para este tipo de topología de red se optó por darle le 1 Gbps en los enlaces troncales con la finalidad de prevenir cuellos de botella y la emulación de redes empresariales y en los enlaces de cada host se empleó 100 Mbps para simular la conexión de clientes realistas, también proporcionar balance entre rendimiento y recursos.

3.7.3. Desarrollo de la aplicación de monitoreo RYU

Al ser la parte central de la red esta aplicación tiene la función de configurar el comportamiento de los conmutadores con relación al tráfico que genera en la red mientras que también presente funciones que análisis ese tráfico y lo registran en un archivo CSV, este tipo de archivos son ideales para entrenar modelos de machine learning [36].

```
def __init__(self, *args, **kwargs):
    super(NetworkMonitor, self).__init__(*args, **kwargs)
    self.mac_to_port = {}
    self.traffic_stats = {}
    self.start_time = time.time()
    self.csv_file = '/home/mininet-v1/ryu/ryu/proyecto/Test-A/A1-respuesta/traffic_log_A.csv'
    self.packet_counter = {}

    # Configurar logging
    self.logger.setLevel(logging.DEBUG)

    with open(self.csv_file, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(["timestamp", "dpid", "src_mac", "dst_mac", "src_ip", "dst_ip",
                         "packet_count", "byte_count", "broadcast_ratio", "mac_change_rate"])
```

Script 6. Código de la ruta de guardad del registro.

El propósito del controlador RYU es ser la parte central de la red el encargado de gestionar y controlar el flujo del tráfico de la red, esta herramienta nos permite configurar de aplicaciones de red en este caso el funcionamiento de la aplicación está pensada para la monitorización, análisis y registro del tráfico de la red en tiempo real. En la captura de los paquetes que pasan por el controlador extrayendo información de ethernet, IPv4 y ARP de forma controlado los recursos de memoria.

En esta parte del código script 6 de la aplicación se especifica la ruta el archivo.

```
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # Capturar todos los paquetes con máxima prioridad
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 65535, match, actions)
```

Script 7. Código de la captura de tráfico.

Para el análisis del tráfico de la red y que sea registrado en los archivos de formato CSV, se aprecia en el script 7. Por otra parte, en el script 8 se ve la ejecución de la aplicación en RYU.

```
› ryu-manager monitor_red_v4.py
```

Script 8. Comando de ejecución de la aplicación de monitore en RYU.

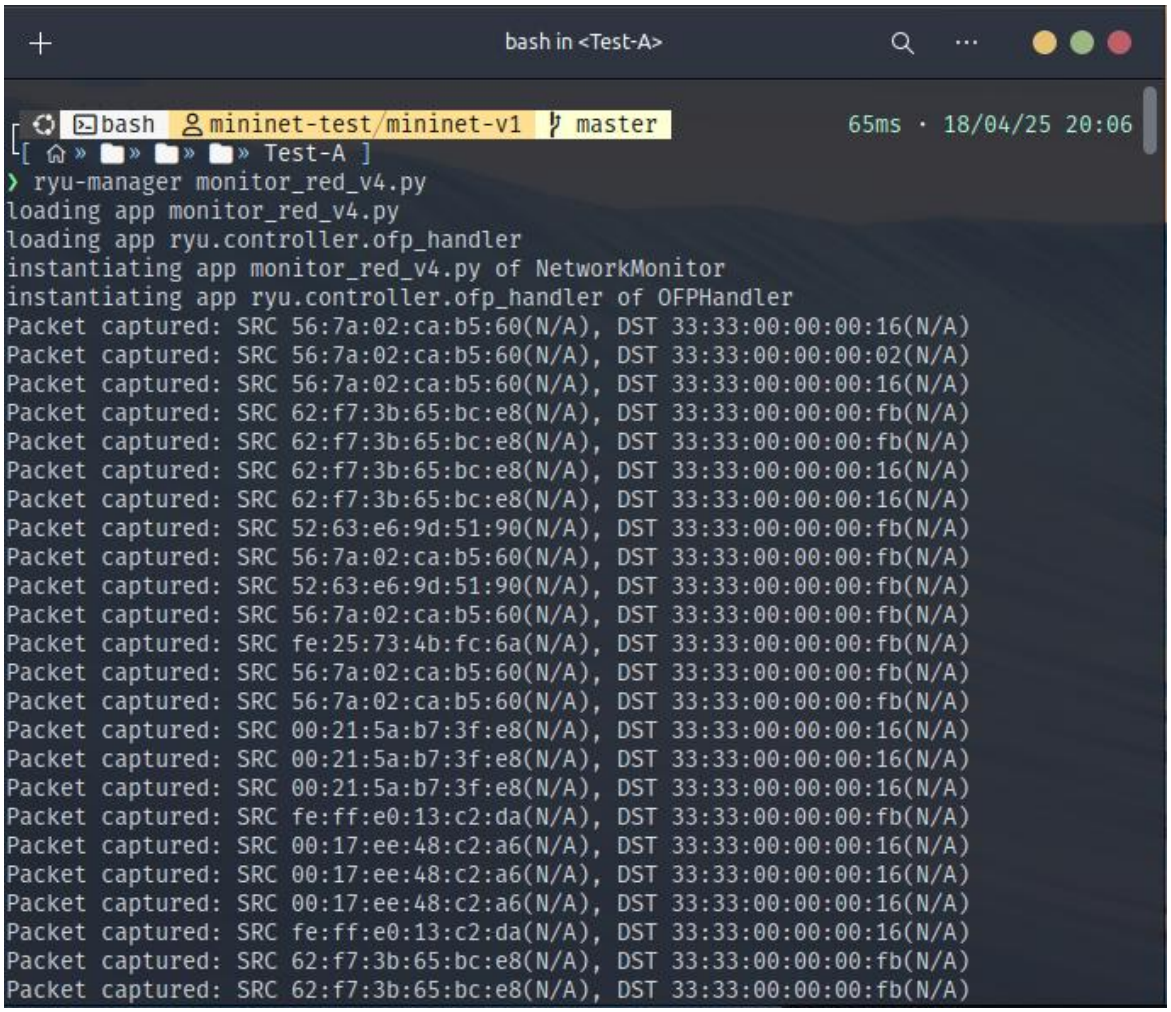


Figura 11. Inicialización de la aplicación de monitoreo de red.

En la consola de la figura 11 se puede apreciar la inicialización del controlador RYU que funciona como un monitor de red con funciones lógicas de switch de capa 2 con relación al funcionamiento de cada uno de los switches de la topología de Mininet.

3.7.4. Generación de tráfico en la red

Esta parte del proyecto es de vital importancia, ya que se requiere mecanismos o formas que nos permitan simular tráfico simulando en entornos reales.

3.7.4.1. Tráfico Normal

Como parte de las pruebas de funcionamiento del tráfico como TCP (figura 12), UDP (figura 13) e ICMP (figura 14) dentro de la red SDN, haciendo uso de las herramientas como 'iperf' y 'ping' para cumplir con el propósito de generar flujo de datos en la red [34].

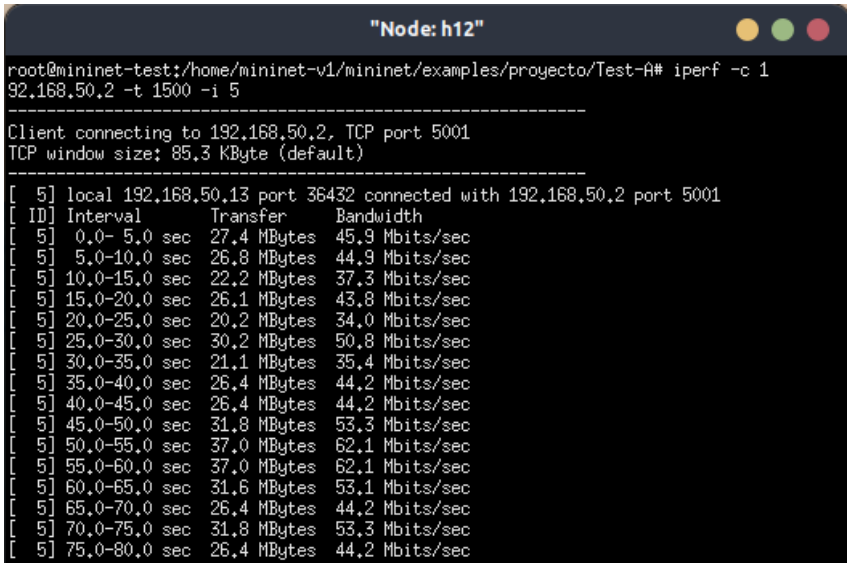


Figura 12. Generación de tráfico TCP.

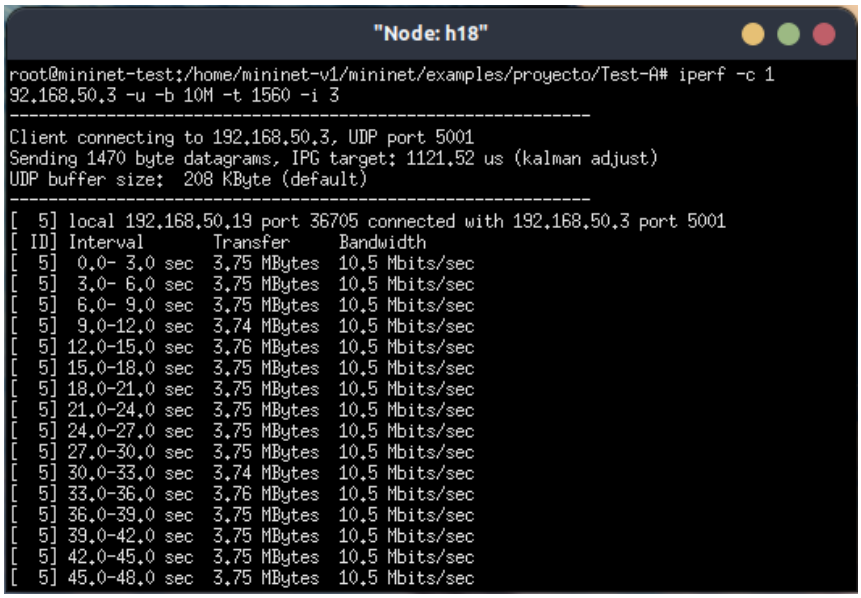


Figura 13. Generación de tráfico UPD.

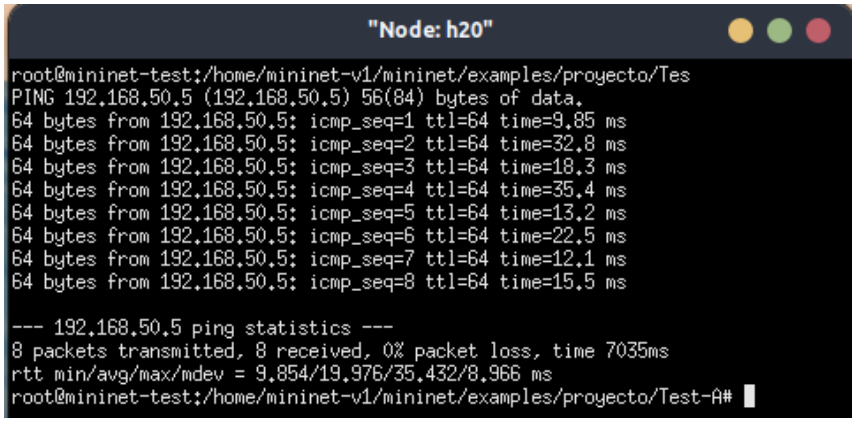


Figura 14. Generación de tráfico ICMP.

3.7.4.2. Tráfico de Ataque

Para el desarrollo del tráfico malicioso es necesario la implementación de un script que genere archivos con MAC aleatorias para generar la simulación con en la figura 15.

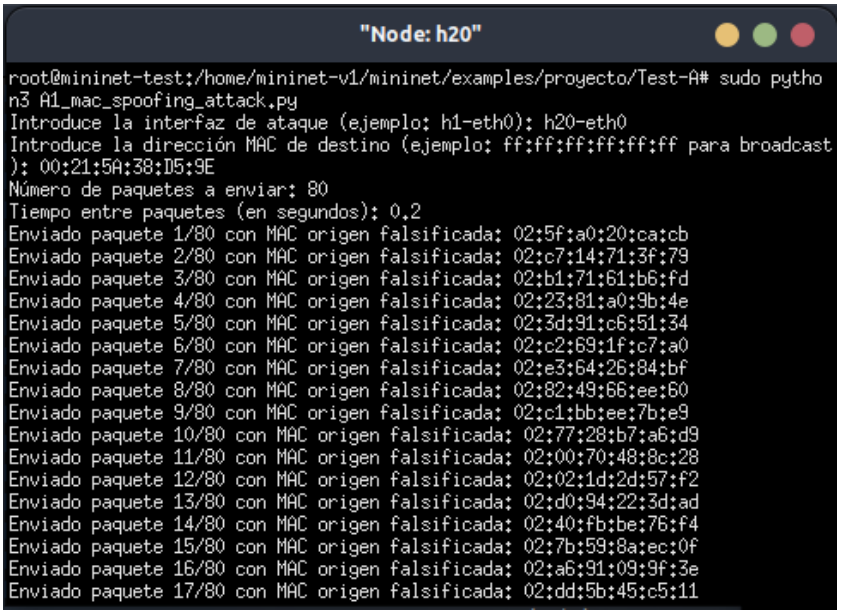


Figura 15. Generación del tráfico malicioso.

3.7.5. DataSet (archivos del tráfico)

Mediante la aplicación diseñada para el controlador RYU, encargada de gestionar el tráfico de la red y registrar todo el tráfico de la red [35]. El Dataset (tabla 7) se forma de todos los datos que pasan por el controlador y se registra considerando las siguientes características:

- Dirección MAC origen/destino.
- Dirección IP origen/destino.
- Frecuencia cambio de MAC origen por host.
- Tiempo entre cambios de MAC.
- Patrones temporales de aparición de nuevas MAC.
- Tasa de paquetes por segundo por MAC.

De forma manual se le asigna una columna en donde se especifica etiquetas para el posterior entrenamiento especificando si es tráfico legitimo (0) o tráfico malicioso (1) [35].

Tabla 7. Estructura del archivo CSV.

Timestamp	dpid	src_mac	dst_mac	src_ip	dst_ip	packet_count	byte_count	broadcast_ratio	mac_change_rate	label
1745034611.10585	33328084990720	00:21:5ª:b7:3f:e8	33:33:00:00:00:16	N/A	N/A	6	110	0	1.180526566652 01	0
1745034867.69192	5042306141184	02:8f:8b:ca:c6:06	00:17:ee:bc:46:f1	N/A	N/A	1	14	0	0.144148378139731	1

3.7.6. Machine Learning (entrenamiento del modelo)

El tipo de modelo de aprendizaje supervisado empleado es la evolución de lo que se considera como árbol de decisiones, el algoritmo 'Random Forest'. También esto es importante si se va a trabajar con el lenguaje Python y la librería Scikit-learn. Para el proceso de entrenamiento es necesario preparar los datos, revisar que esta información está homogénea, la etiquetación para especificar qué tipo de tráfico se trata [33].

Librerías implementadas para el desarrollo del modelo entrenado se ve en el script 9.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
```

Script 9. Encabezado del código para el entrenamiento.

Para emplear el algoritmo se consideró el 70% de entrenamiento y 30% de pruebas, como se ve en el script 10. La validación cruzada se visualiza en los anexos 4.

```
def train_model(data_path, model_path):
    """
    Entrena el modelo Random Forest Classifier y lo guarda en un archivo
    """
    # Cargar los datos
    print(f"Cargando datos desde {data_path}...")
    df = pd.read_csv(data_path)

    # Mostrar información básica del dataset
    print("\n=== Información del Dataset ===")
    print(f"Dimensiones: {df.shape}")
    print(f"Columnas: {df.columns.tolist()}")
    print("\nDistribución de clases:")
    if 'label' in df.columns:
        class_distribution = df['label'].value_counts(normalize=True) * 100
        print(class_distribution)
        print(f"Número de muestras normales (0): {df[df['label']==0].shape[0]}")
        print(f"Número de muestras de ataque (1): {df[df['label']==1].shape[0]}")
    else:
        print("Error: Columna 'label' no encontrada en el dataset")
        return
```

Script 10. Código de entrenamiento del algoritmo.

```
❯ python3 RFC_ml_model.py
```

Script 11. Comando de ejecución del algoritmo.

Para ejecutar el algoritmo (script 11) es necesario especificar la versión de Python se va a emplear.

3.7.7. Aplicación RYU de detección

Una vez ya obtenido el modelo de entrenamiento, es necesario emplear una aplicación que efectué el uso de este modelo, para que analice el tráfico de la red y, dependiendo a los patrones ya entrenados, comience a alertar sobre el tipo de ataque como se ve en el código de script 12 y la inicialización de la aplicación (script 13), al efectuarse en la red SDN [50].

```
# Preparar características para el modelo
        features = self.prepare_features(pkt_data)
                # Alerta de ataque detectado
                self.logger.warning(f"¡ALERTA! Posible ataque Spoofing MAC detectado:")
                self.logger.warning(f"MAC: {src_mac} IP: {src_ip} Puerto: {in_port} Confianza: {confidence:.2%}")
                # Registrar el ataque en CSV
                with open(self.csv_file, 'a', newline='') as f:
                    writer = csv.writer(f)
                    writer.writerow([
                        pkt_data["timestamp"],
                        dpid,
                        src_mac,
                        dst_mac,
                        src_ip,
                        dst_ip,
                        pkt_data["packet_count"],
                        pkt_data["byte_count"],
                        broadcast_ratio,
                        mac_change_rate,
                        confidence,
                        "MAC_Spoofing"
                    ])
```

Script 12. Código del apartado de aplicación en RYU para la alerta.

```
❯ ryu-manager detector_RYU-6.py
```

Script 13. Comando de ejecución de la aplicación que detecta en RYU.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

En esta sección, se expone los datos obtenidos mediante el análisis estadístico, para proporcionar la evidencia que respalda la investigación.

4.1. Resultados

En este estudio la orientación del modelo es a la selección y funcionalidad optima.

4.1.1. Rendimiento de la máquina virtual

Dentro de la máquina virtual al momento de en estado tranquilo sin ninguna interacción de aplicaciones presenta un consumo de la CPU que varía entre 1.3% hasta un 4.8% al momento de desplegar la red SDN conformada por 5 switches, 22 hosts y la generación del tráfico con un incremento de los recursos de la CPU variando entre 26.9% hasta un 32.6%, para el trascurso de las pruebas (figura 16).

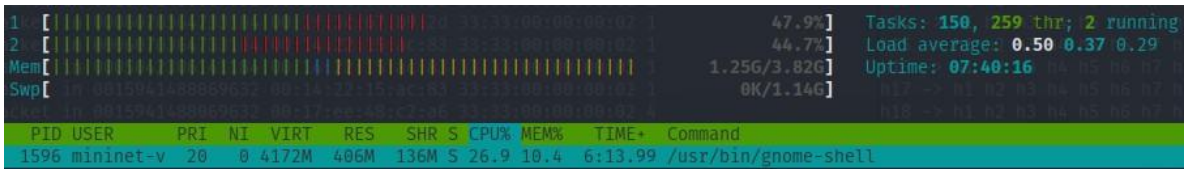


Figura 16. Consumo de la CPU mientras se realiza las pruebas en Mininet.

En el apartado del consumo del recurso de la memoria RAM que presenta antes de generar las pruebas tiene 1.19 GB y al momento de desplegar las actividades del proyecto, el incremento es de 1.25 GB de uso del sistema (figura 16).

4.1.2. Tráfico Obtenido

En la gráfica del tráfico (figura 17) de la red, en los puntos más altos tiene relación sobre los hosts que están generando todo el tráfico incluyendo los ataques dentro de la red SDN [50].

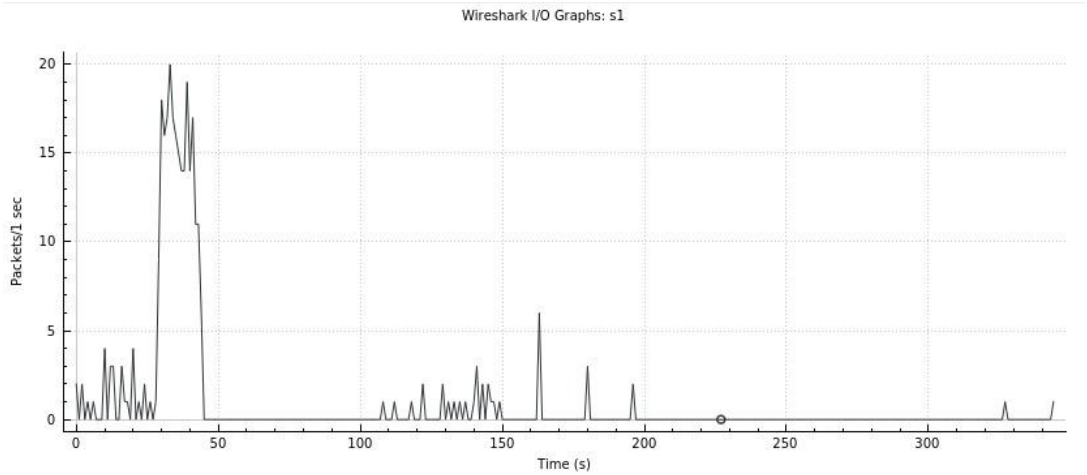


Figura 17. Tráfico de la red SDN en el proceso de pruebas.

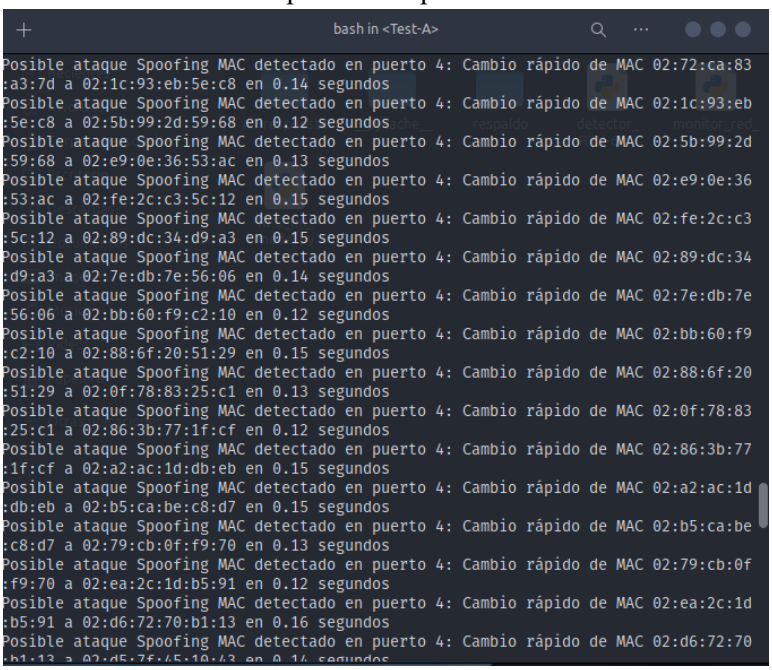


Figura 18. Implementación del modelo de seguridad.

En la figura 18 se puede observar las alertas por consola de la aplicación de seguridad al ponerla aprueba dentro del entorno controlado.

4.1.3. La precisión obtenida de los modelos generados

Los modelos entrenados fueron tres modelos entrenados a partir de Dataset, cada uno de ellos con diferente cantidad de tráfico registrado. Para el primer modelo se generó un registro de 500, con el segundo se tiene 800, el último 1100 registros.
Dentro de la configuración del código para la segmentación de Dataset en 70% para entrenamiento y 30% para test [51].

Modelo 1

El modelo 1, al contar con una menor cantidad de datos especificados de ataque porque muy pocos de los nodos u hosts son los que están generando tráficos, en la detección de RYU presenta una mayoría de fallas en alertar el ataque (tabla 8).

Modelo 2

Para el modelo 2, ya que se incrementa el número de datos registrados a 800, presenta menos incongruencias que el modelo anterior, en este entrenamiento se incorporó un mayor número de nodos que se encargan de la generación del tráfico de la red (tabla 8).

Modelo 3

El modelo 3, el cual, presenta una mayor robustez que sus modelos predecesores, ya que la cantidad de datos incorporados es aún mayor, por el hecho de que se incrementó el número de nodos que permitieron una mayor carga de flujo de datos para las pruebas (tabla 8).

Tabla 8. Precisión presente por los modelos empleando los DataSet para entrenar.

Modelo	Precisión	Datos Registrados
1	67%	500
2	84 %	800
3	92%	1100

En el proceso de testeo de la aplicación en RYU para detectar el tráfico malicioso, se recopilaron 200 salidas de datos para cada modelo y permitió la verificación de cada modelo de detección. Se presenta verdadero, positivo (detecta ataque) o negativo (detecta nada) y también falsos positivos (detecta tráfico normal como ataque) o negativos (no detecta el ataque) indicando en la tabla 9. Se realizó un ataque aproximado de 300 ataques

(anexo 3) y un tráfico normal aproximado de 50 paquetes, entre cada uno de los hosts seleccionados.

Tabla 9. Los valores presentados en cada caso sobre verdades positivos y negativos, falsos positivos y negativos, también la correspondiente precisión.

Modelo	Falso Positivos	Falsos Negativos	Verdadero Positivo	Verdadero Negativo	Precisión
1	31	18	4	14	67%
2	43	29	1	11	84%
3	62	23	0	7	92%

En la siguiente figura 19 se presenta la precisión de los modelos de una forma gráfica, evidenciando que el último modelo presenta una mayor precisión.

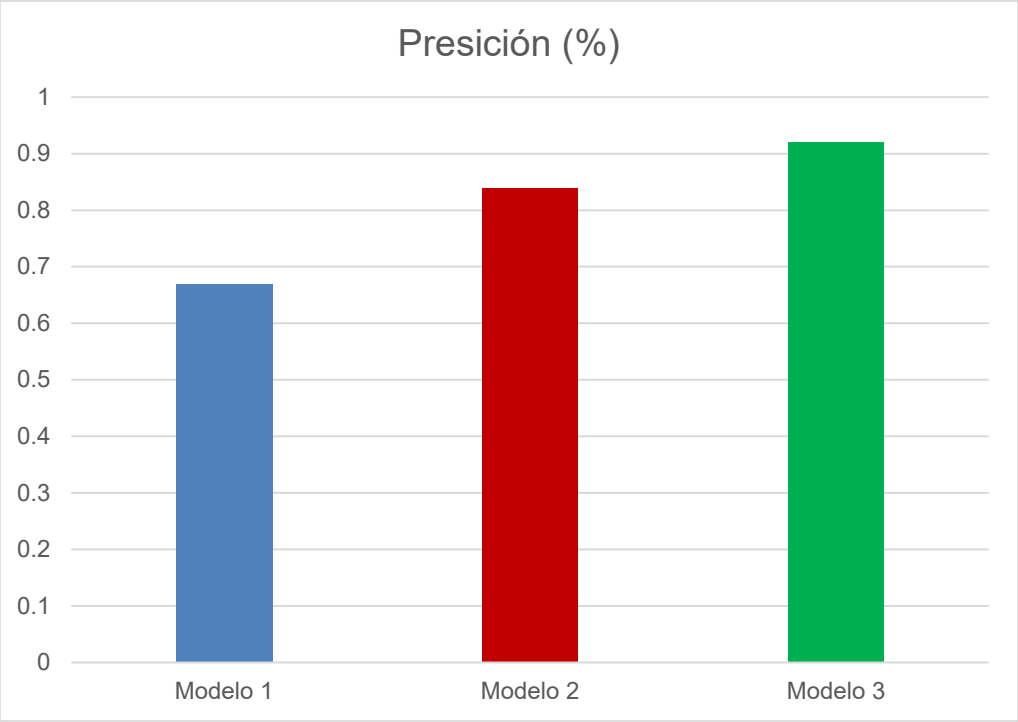


Figura 19. Gráfica de la precisión de los modelos.

4.1.4. Prueba de hipótesis

Prueba de Kruskal-Wallis

Para este proyecto se aplicó la prueba Kruskal-Wallis el cual está pensado para la comparación entre tres o más grupos independientes, en este caso analizar la diferencia del rendimiento de modelos entrenados con Dataset de 500, 800 y 1100 muestras.

Planteamiento de la hipótesis

H_0: La distribución de precisión en el número de ataques detectados es la misma en los 3 modelos.

H_a: La distribución de precisión en el número de ataques detectados es diferente en al menos uno de los 3 modelos.

Tabla 10. Síntesis de Hipótesis.

Resumen de contrastes de hipótesis				
	Hipótesis nula	**Prueba**	**Sig.**	**Decisión**
1	La distribución de precisión en el número de ataques detectados es la misma en los 3 modelos.	Prueba de Kruskal-Wallis para muestra independientes.	0.000	Se rechaza la hipótesis nula.
Se muestra significaciones asintóticas. El nivel de significación es de 0,050.				

Tabla 11. Test no paramétrico de Kruskal-Wallis

Test Kruskal-Wallis					
Modelo	**Precisión (medianas)**	**Nivel de significancia**	**Estadística de prueba**	**P-valor (Sig)**	**Grado de libertad**
1	67 %	α = 0,05	chi-squared = 25,98	0,00	2
2	84 %				
3	92 %				

Interpretación:

- De acuerdo con el valor visto en las medianas el modelo 3 (entrenado con 1100 datos) presenta una mayor respuesta en relación con la alerta de ataques.

- Para el sig o p-valor '0,00' nos da a entender que en apartado estadístico existe una diferencia significativa entre los modelos entrenados: de esta forma se rechaza la hipótesis nula y se acepta la hipótesis alternativa (tabla 10) ya que se entiende que p-valor < α.
- En relación con el contenido de la tabla 11, se presenta una diferencia significativa en la precisión de cada uno de los modelos en la alerta de amenazas, con un nivel de confianza de 95 % en este análisis.

En definitiva, la prueba Kruskai-Wallis indica diferencias significativas en la precisión de detección de los ataques para cada modelo, ya que en la figura 20, los modelos 1, 2, 3, con unas medias de 92%, 84%, 67%, esto nos indica que el modelo 3 se considera como el que presenta mejores resultados.

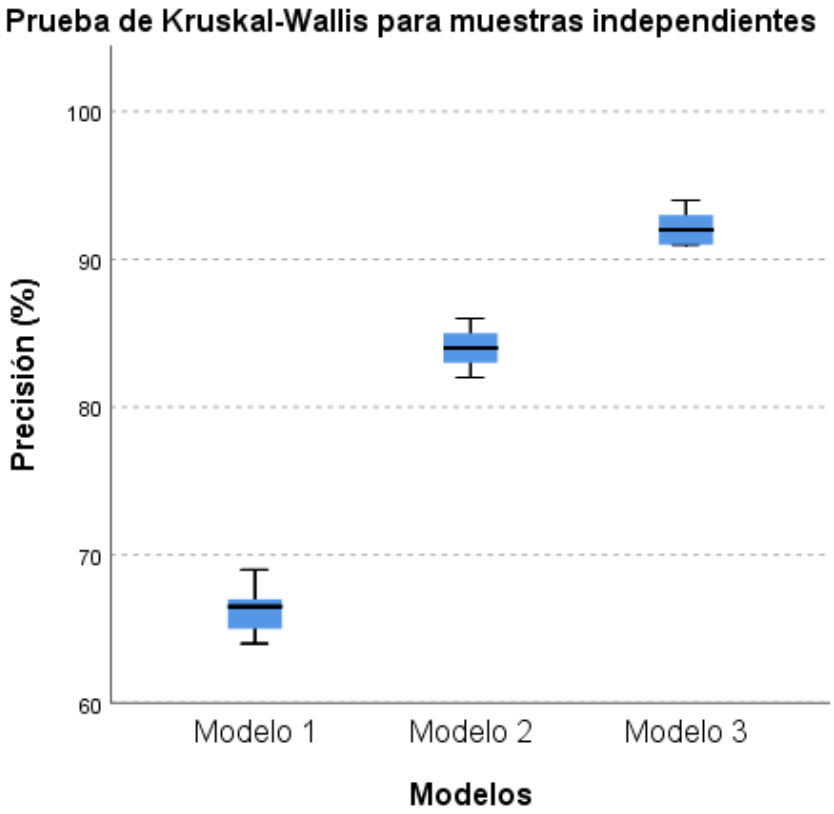


Figura 20. Muestra Independiente (Kruskal-Wallis).

CAPÍTULO V

CONCLUSIONES y RECOMENDACIONES

En esta sección, se centra en el análisis y el resumen de los resultados, proporcionado una orientación para que otras investigaciones en el futuro.

5.1. Conclusión

En el presente proyecto de titulación se realizó la implementación de un modelo de seguridad basado en técnicas de machine learning, planificado para la arquitectura de redes definidas por software (SDN), tomando en cuenta las herramientas de código abierto accesibles para el desarrollo del proyecto. Herramientas como Mininet y RYU entre otras, fueron revisadas y evaluadas para poder comprobar la compatibilidad dentro del entorno de pruebas. El modelo de seguridad combina el algoritmo supervisado y la mejor versión de modelo presenta un 92% de precisión sobre los conjuntos de pruebas, dando a entender la importancia de estos mecanismos de seguridad sobre las redes SDN.

La investigación ayudó con la identificación de las amenazas más comunes tanto de las redes convencionales como las arquitecturas de redes SDN, sobre todo ataques al plano de control, ataques en la capa de comunicación entre los switches. Para este caso se abordaron los ataques MAC Spoofing. Se concluyó que este tipo de ataques afecta a la integridad y compromiso de los datos, presentando robo de la información, mientas que los ataques al plano de control generan una ralentización de la red SDN en un aproximado de 28%.

Tras la evaluación entre diversos algoritmos de machine learning, se determinó que Random Forest proporciona el mejor equilibrio entre la precisión y rendimiento para la detección de anomalías, cuando se despliega el tráfico en las redes SDN. Por otra parte, se adaptó este algoritmo para procesar las características necesarias para detectar el vector de ataque dentro del flujo OpenFlow. La mayoría de soluciones están enfocadas a la detección de ataques DDoS. Para el caso de nuestro vector de ataque, los modelos entrenados presentes conforman un mayor entrenamiento y tienen menor reducción de los falsos positivos.

Se implementó un conjunto de herramientas que permitieron satisfactoriamente un sistema de alerta al trabajar con módulos personalizados en Python para el procesamiento de datos del tráfico de la red SDN. Igualmente, la implementación aprovecha las API nativas para los controles open source como RYU, tanto en el apartado de la recolección de datos para generar los DataSet en tiempo real, así como la identificación del tráfico malicioso con el modelo entrenado.

Para las pruebas realizadas en un entorno controlado, al simular una red convencional de mediana escala con 5 switches y 22 hosts virtualizados al combinar con una aplicación de recolección y detección para el controlador RYU implementando, agregando el modelo entrenado, el sistema presento una precisión del 92%. La evaluación presentaba para el prototipo del sistema, particularmente efectivo en la detección de vectores de ataque MAC Spoofing identificando en promedio de 28 segundos.

De acuerdo con las limitaciones y características del dispositivo anfitrión y la máquina virtual como el entorno seguro para cada una de las pruebas del prototipo de seguridad al emplear cada uno de los modelos entrenados y la cantidad de ataques generado, optando por 300 ataques como el máximo realizado en los experimentos del modelo de seguridad.

5.2. Recomendaciones

Para pruebas de mayor escala, con topologías más complejas o generación de múltiples vectores de ataque y tener un tráfico mucho más complejo, sería mucho más conveniente que tener un dispositivo anfitrión de mayores características para pruebas más sofisticadas.

Otros de los factores importantes es tener buenas prácticas de configuración en el desarrollo de la arquitectura de redes centralizadas así como usar un implementar la mejor opción en cuanto a la gestión del controlador, con la correcta configuración de las aplicaciones encargadas del comportamiento de los enrutadores y del flujo de tráfico de la red.

Tomar en cuenta que para las versiones más modernas del sistema Ubuntu, ciertas aplicaciones usadas en este proyecto no eran compatibles de acuerdo con las distribuciones de la rama LTS, al presentar bugs y poca documentación que permita el despliegue de ciertas herramientas, optando por trabajar en una única versión especificada con anterioridad.

Para futuros trabajos, que empleen las redes definidas por software enfocado en investigaciones que desarrollen técnicas de optimización o detección de fallas con relación a malas configuraciones de la red o del funcionamiento del controlador, también sería muy interesante ver proyectos que usen sistemas de seguridad de mayor complejidad y optimizados en la detección o altera de diferentes tipos de ataques penados para dispositivos de mayor potencia.

BIBLIOGRAFÍA

[1] c. Kodzai y j. Mwangama, “impact of network security on the sdn controller performance”, 2020.

[2] m. P. Han, “hybrid gns3 and mininet-wifi emulator for survivable sdn hybrid gns3 and mininet-wifi emulator for survivable sdn backbone network supporting wireless iot traffic backbone network supporting wireless iot traffic”. [en línea]. Disponible en: https://digital.car.chula.ac.th/chulaetd

[3] n. Gupta, m. S. Maashi, s. Tanwar, s. Badotra, m. Aljebreen, y s. Bharany, “a comparative study of software defined networking controllers using mininet”, el 1 de septiembre de 2022, mdpi. Doi: 10.3390/electronics11172715.

[4] a. Sai, m. Shivakumar, y y. Jabbu, “master of science in telecommunication systems securing sdn data plane: investigating the effects of ip spoofing attacks on sdn switches and its mitigation simulation of ip spoofing using mininet”, 2023. [en línea]. Disponible en: www.bth.se

[5] m. N. A. Sheikh, i. S. Hwang, m. S. Raza, y m. S. Ab-rahman, “a qualitative and comparative performance assessment of logically centralized sdn controllers via mininet emulator”, el 1 de abril de 2024, multidisciplinary digital publishing institute (mdpi). Doi: 10.3390/computers13040085.

[6] e. L. Z. N. Min wei, 2020 international conference on information networking (icoin). Ieee, 2020.

[7] dr. P. B. Patil*, mr. K. S. Bhagat., dr. D. K. Kirange, y dr. S. D. Patil, “software defined networks using mininet”, international journal of recent technology and engineering (ijrte), vol. 9, núm. 1, pp. 843–849, may 2020, doi: 10.35940/ijrte.f9375.059120.

[8] w. A. Alonazi, h. Hamdi, n. A. Azim, y a. A. Abd el-aziz, “sdn architecture for smart homes security with machine learning and deep learning”. [en línea]. Disponible en: www.ijacsa.thesai.org

[9] d. Y. Setiawan, s. N. Hertiana, y r. M. Negara, “6lowpan performance analysis of iot software-defined-network-based using mininet-io”, en iotais 2020 - proceedings: 2020 ieee international conference on internet of things and intelligence systems, institute of electrical and electronics engineers inc., ene. 2021, pp. 60–65. Doi: 10.1109/iotais50849.2021.9359714.

[10] h. M. Noman y m. N. Jasim, “pox controller and open flow performance evaluation in software defined networks (sdn) using mininet emulator”, en iop conference series: materials science and engineering, institute of physics publishing, ago. 2020. Doi: 10.1088/1757-899x/881/1/012102.

[11] d. Cabarkapa y d. Rancic, “performance analysis of ryu-pox controller in different tree-based sdn topologies”, advances in electrical and computer engineering, vol. 21, núm. 3, pp. 31–38, 2021, doi: 10.4316/aece.2021.03004.

[12] p. Pandey y r. Kumar sah, “proceedings of 8 th ioe graduate conference security analysis of sdn using blockchain technology”.

[13] o. I. Romanov, i. O. Saychenko, a. I. Marinov, y s. S. Skolets, “research of sdn network performance parameters using mininet network emulator”, 2020.

[14] p. Zhang, “performance analysis of floodlight and ryu sdn controllers under mininet simulator 1 st yanzhen li 4 th bo peng 5 th xiaoyue li”, 2020.

[15]. Z askarinejadamiri, n. Nourani, y n. Zanjanicskandari, “comparative study of data transfer in sdn network architecture in iot”, international journal of web research, vol. 6, núm. 1, pp. 95–104, 2023, doi: 10.22133/ijwr.2024.421267.1186.

[16] v. T. Noora y s. Vinila jinny, “design of different topologies in sdn controller”, indian journal of computer science and engineering, vol. 11, núm. 5, pp. 582–592, sep. 2020, doi: 10.21817/indjcse/2020/v11i5/201105203.

[17] m. M. Cherian y s. L. Varma, “mitigation of ddos and mitm attacks using belief based secure correlation approach in sdn-based iot networks”, international journal of computer network and information security, vol. 14, núm. 1, pp. 52–68, feb. 2022, doi: 10.5815/ijcnis.2022.01.05.

[18] m. Christensson y e. De paz, “enhancing internet security and mobility with host identity pro-tocol: integration, testing, and optimization-simulation of hip-vpls using mininet förbättring av säkerhet och rörlighet på internet genom host identity protocol: integration, testning och optimering”. [en línea]. Disponible en: www.liu.se

[19] m. Hasan, h. Dahshan, e. Abdelwanees, y a. Elmoghazy, “sdn mininet emulator benchmarking and result analysis”, en 2nd novel intelligent and leading emerging sciences conference, niles 2020, institute of electrical and electronics engineers inc., oct. 2020, pp. 355–360. Doi: 10.1109/niles50944.2020.9257913.

[20] a. K. A. Al-mashadani y m. Ilyas, “distributed denial of service attack alleviated and detected by using mininet and software defined network”, webology, vol. 19, núm. 1, pp. 4129–4144, ene. 2022, doi: 10.14704/web/v19i1/web19272.

[21] p. Khantirava et al., “early detection and prevention of distributed denial of service attack using software-defined networks with mininet network simulator”, international research journal of engineering and technology, 2023, [en línea]. Disponible en: www.irjet.net

[22] p.-t. Tivig y e. Borcoci, “performance evaluation experiments for video and voip traffic with ryu controller and mininet framework-part i”, u.p.b. sci. Bull., series c, vol. 84, núm. 3, p. 2022.

[23] p. R. Grammatikis et al., “sdn-based resilient smart grid: the sdn-microsense architecture”, digital, vol. 1, núm. 4, pp. 173–187, dic. 2021, doi: 10.3390/digital1040013.

[24] d. Kumar y m. Sood, “software defined networks (s.d.n): experimentation with mininet topologies”, indian j sci technol, vol. 9, núm. 32, ago. 2016, doi: 10.17485/ijst/2016/v9i32/100195.

[25] l. M. Amaya fariño, j. F. Arroyo pizarro, m. Jaramillo infante, a. R. Tumbaco reyes, y b. M. Mendoza morán, “sdn redes definidas por software usando mininet”, revista científica y tecnológica upse, vol. 9, núm. 1, pp. 48–56, jun. 2022, doi: 10.26423/rctu.v9i1.489.

[26] g. Florance y r. J. Anandhi, “study on sdn with security issues using mininet”, advances in parallel computing, vol. 39, pp. 104–113, 2021, doi: 10.3233/apc210186.

[27] m. G. Spina, m. Tropea, y f. De rango, “mitigation of lldp topological poisoning attack in sdn environments using mininet emulator”, en proceedings of the international conference on simulation and modeling methodologies, technologies and applications, science and technology publications, lda, 2023, pp. 318–325. Doi: 10.5220/0012086200003546.

[28] d. Hamad, k. Yalda, n. Tapus, y i. T. Okumus, “enhancing iot scalability and security through sdn”, revista română de informatică și automatică, vol. 34, núm. 2, pp. 113–126, jun. 2024, doi: 10.33436/v34i2y202409.

[29] m. Deris ramdhani, b. Sugiarto, y a. Rukmana, “simulasi jaringan sdn menggunakan controller ryu pada mininet dengan 5 topologi jaringan”, jurnal fuse-teknik elektro |vol. 1 |, núm. 2.

[30] n. Satheesh et al., “flow-based anomaly intrusion detection using machine learning model with software defined networking for openflow network”, microprocess microsyst, vol. 79, nov. 2020, doi: 10.1016/j.micpro.2020.103285.

[31] s. Alessio, a. Mardaus, e. Biernacka, r. Wójcik, y j. Dom˙, “open source software-defined networking controllers-operational and security issues”, 2024, doi: 10.3390/electronics.

[32] p. Escudero, w. Alcocer, y j. Paredes, “recurrent neural networks and arima models for euro/dollar exchange rate forecasting”, applied sciences (switzerland), vol. 11, núm. 12, jun. 2021, doi: 10.3390/app11125658.

[33] k. Nisar et al., “a survey on the architecture, application, and security of software defined networking: challenges and open issues”, el 1 de diciembre de 2020, elsevier b.v. doi: 10.1016/j.iot.2020.100289.

[34] m. Hussain, n. Shah, r. Amin, s. S. Alshamrani, a. Alotaibi, y s. M. Raza, “software-defined networking: categories, analysis, and future directions”, sensors, vol. 22, núm. 15, ago. 2022, doi: 10.3390/s22155551.

[35] w. A. Alonazi, h. Hamdi, n. A. Azim, y a. A. A. El-aziz, “sdn architecture for smart homes security with machine learning and deep learning”, international journal of advanced computer science and applications, vol. 13, núm. 10, pp. 917–927, 2022, doi: 10.14569/ijacsa.2022.01310108.

[36] c. Guerber, m. Royer, y n. Larrieu, “machine learning and software defined network to secure communications in a swarm of drones”, journal of information security and applications, vol. 61, sep. 2021, doi: 10.1016/j.jisa.2021.102940.

[37] p. Zhang, “performance analysis of floodlight and ryu sdn controllers under mininet simulator 1 st yanzhen li 4 th bo peng 5 th xiaoyue li”, 2020.

[38] e. Gurjão, k. Dias, y m. Amaris, “tolerância a falhas em multi-controladores sdn: eficiência de controladores ryu usando mininet”.

[39] p. Pandey y r. Kumar sah, “proceedings of 8 th ioe graduate conference security analysis of sdn using blockchain technology”.

[40] antonio agustín pastor perales, “antonio_agustin_pastor_perales”, 2022.

[41] m. Christensson y e. De paz, “enhancing internet security and mobility with host identity pro-tocol: integration, testing, and optimization-simulation of hip-vpls using mininet förbättring av säkerhet och rörlighet på internet genom host identity protocol: integration, testning och optimering”. [en línea]. Disponible en: www.liu.se

[42] p. Khantirava et al., “early detection and prevention of distributed denial of service attack using software-defined networks with mininet network simulator”, international research journal of engineering and technology, 2023, [en línea]. Disponible en: www.irjet.net

[43] p.-t. Tivig y e. Borcoci, “performance evaluation experiments for video and voip traffic with ryu controller and mininet framework-part i”, u.p.b. sci. Bull., series c, vol. 84, núm. 3, p. 2022.

[44] a. K. Jaiswal, “dos attack network traffic monitoring in software defined networking using mininet and ryu controller”, el 8 de diciembre de 2022. Doi: 10.21203/rs.3.rs-2282189/v1.

[45] j. Cunha et al., “enhancing network slicing security: machine learning, software-defined networking, and network functions virtualization-driven strategies”, future internet, vol. 16, núm. 7, jul. 2024, doi: 10.3390/fi16070226.

[46] p. Karthika y k. Arockiasamy, “simulation of sdn in mininet and detection of ddos attack using machine learning”, bulletin of electrical engineering and informatics, vol. 12, núm. 3, pp. 1797–1805, jun. 2023, doi: 10.11591/eei.v12i3.5232.

[47] j. Bhayo, s. A. Shah, s. Hameed, a. Ahmed, j. Nasir, y d. Draheim, “towards a machine learning-based framework for ddos attack detection in software-defined iot (sd-iot) networks”, eng appl artif intell, vol. 123, ago. 2023, doi: 10.1016/j.engappai.2023.106432.

[48] dylan da costa, omkar pradip naik, y omkar randeep pawar, “towards effective intrusion detection in openflow-based sdn architectures”, international journal of scientific research in computer science, engineering and information technology, pp. 210–220, dic. 2023, doi: 10.32628/cseit2390631.

[49] h. Polat, o. Polat, y a. Cetin, “detecting ddos attacks in software-defined networks through feature selection methods and machine learning models”, sustainability (switzerland), vol. 12, núm. 3, feb. 2020, doi: 10.3390/su12031035.

[50] ö. Tonkal, h. Polat, e. Başaran, z. Cömert, y r. Kocaoğlu, “machine learning approach equipped with neighbourhood component analysis for ddos attack detection in software-defined networking”, electronics (switzerland), vol. 10, núm. 11, jun. 2021, doi: 10.3390/electronics10111227.

[51] t. A. Tang, l. Mhamdi, d. Mclernon, s. A. R. Zaidi, m. Ghogho, y f. El moussa, “deepids: deep learning approach for intrusion detection in software defined networking”, electronics (switzerland), vol. 9, núm. 9, pp. 1–18, sep. 2020, doi: 10.3390/electronics9091533.

ANEXOS

Anexo 1. Códigos extras

En esta parte se encuentra todo códigos necesarios en el desarrollo de las herramientas necesarias para el despliegue del proyecto.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSSwitch
from mininet.link import TCLink
from mininet.log import setLogLevel, info
from mininet.cli import CLI

def customNetwork():
    net = Mininet(controller=RemoteController, switch=OVSSwitch, link=TCLink)

    info("*** Añadir controlador\n")
    controller = net.addController("c0", controller=RemoteController, ip="127.0.0.1", port=6633)

    info("*** Añadir switches\n")
    s1 = net.addSwitch("s1", cls=OVSSwitch,protocols='OpenFlow13', dpid='00001A2FBBCCDD00')
    s2 = net.addSwitch("s2", cls=OVSSwitch,protocols='OpenFlow13', dpid='00000E7FAABBCC00')
    s3 = net.addSwitch("s3", cls=OVSSwitch,protocols='OpenFlow13', dpid='000005855DEEFF00')
    s4 = net.addSwitch("s4", cls=OVSSwitch,protocols='OpenFlow13', dpid='00001E4FCCAABB00')
    s5 = net.addSwitch("s5", cls=OVSSwitch,protocols='OpenFlow13', dpid='0000049600DDCC00')
```

```
info("*** Añadir hosts\n")
    # Hosts conectados a switch 2
    h1 = net.addHost("h1", ip="192.168.50.2/24", defaultRoute="via 192.168.50.1",
mac="00:25:B3:64:8E:25")
    h2 = net.addHost("h2", ip="192.168.50.3/24", defaultRoute="via 192.168.50.1",
mac="00:21:5A:B7:3F:E8")
    h3 = net.addHost("h3", ip="192.168.50.4/24", defaultRoute="via 192.168.50.1",
mac="00:17:EE:48:C2:A6")

    # Hosts conectados a switch 3
    h4 = net.addHost("h4", ip="192.168.50.5/24", defaultRoute="via 192.168.50.1",
mac="00:14:22:5D:91:BF")
    h5 = net.addHost("h5", ip="192.168.50.6/24", defaultRoute="via 192.168.50.1",
mac="00:1F:3C:E3:7A:D4")
    h6 = net.addHost("h6", ip="192.168.50.7/24", defaultRoute="via 192.168.50.1",
mac="04:92:26:89:BE:5C")
    h7 = net.addHost("h7", ip="192.168.50.8/24", defaultRoute="via 192.168.50.1",
mac="00:25:B3:FA:C4:17")
    h8 = net.addHost("h8", ip="192.168.50.9/24", defaultRoute="via 192.168.50.1",
mac="00:21:5A:38:D5:9E")
    h9 = net.addHost("h9", ip="192.168.50.10/24", defaultRoute="via 192.168.50.1",
mac="00:17:EE:BC:46:F1")

# Hosts conectados a switch 4
    h10 = net.addHost("h10", ip="192.168.50.11/24", defaultRoute="via 192.168.50.1",
mac="00:14:22:15:AC:83")
    h11 = net.addHost("h11", ip="192.168.50.12/24", defaultRoute="via 192.168.50.1",
mac="00:1F:3C:67:B9:2D")
    h12 = net.addHost("h12", ip="192.168.50.13/24", defaultRoute="via 192.168.50.1",
mac="04:92:26:D2:93:EF")
    h13 = net.addHost("h13", ip="192.168.50.14/24", defaultRoute="via 192.168.50.1",
mac="00:25:B3:41:8C:76")
```

```
# Hosts conectados a switch 5
    h18 = net.addHost("h18", ip="192.168.50.19/24", defaultRoute="via 192.168.50.1",
mac="04:92:26:75:31:AB")
    h19 = net.addHost("h19", ip="192.168.50.20/24", defaultRoute="via 192.168.50.1",
mac="00:25:B3:9C:45:E1")
    h20 = net.addHost("h20", ip="192.168.50.21/24", defaultRoute="via 192.168.50.1",
mac="00:21:5A:19:23:DC")
    h21 = net.addHost("h21", ip="192.168.50.22/24", defaultRoute="via 192.168.50.1",
mac="00:17:EE:54:1F:8B")
    h22 = net.addHost("h22", ip="192.168.50.23/24", defaultRoute="via 192.168.50.1",
mac="00:14:22:A7:B3:1C")

    info("*** Creación de enlaces\n")
    # Conexión con de los switches con el switch central - 1Gbps con 0.5ms de delay
    net.addLink(s1, s2, cls=TCLink, bw=1000, delay='0.5ms')
    net.addLink(s1, s3, cls=TCLink, bw=1000, delay='0.5ms')
    net.addLink(s1, s4, cls=TCLink, bw=1000, delay='0.5ms')
    net.addLink(s1, s5, cls=TCLink, bw=1000, delay='0.5ms')

    # Conexión de los hosts al switch 2 - 100Mbps con 1ms de delay
    net.addLink(h1, s2, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h2, s2, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h3, s2, cls=TCLink, bw=100, delay='1ms')

# Conexión de los hosts al switch 3 - 100Mbps con 1ms de delay
    net.addLink(h4, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h5, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h6, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h7, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h8, s3, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h9, s3, cls=TCLink, bw=100, delay='1ms')
```

```
# Conexión de los hosts al switch 4 - 100Mbps con 1ms de delay
    net.addLink(h10, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h11, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h12, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h13, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h14, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h15, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h16, s4, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h17, s4, cls=TCLink, bw=100, delay='1ms')

    # Conexión de los hosts al switch 5 - 100Mbps con 1ms de delay
    net.addLink(h18, s5, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h19, s5, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h20, s5, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h21, s5, cls=TCLink, bw=100, delay='1ms')
    net.addLink(h22, s5, cls=TCLink, bw=100, delay='1ms')

    info("*** Inicializar Red\n")
    net.start()
    s1.cmd("ifconfig s1 192.168.50.1/24")

    info("*** Ejecutando CLI\n")
    CLI(net)

    info("*** Finalizar Red\n")
    net.stop()

if __name__ == "__main__":
    setLogLevel("info")
    customNetwork()
```

Script 14. Código de la topología de la red en Mininet.

Anexo 2. Datos recopilados

Esta parte se encuentra los datos que se recopilaron en los dataset.

	Predeterminado	Predeterminado	Predeterminado	Predeterminado	Predeterr	Predeterr	Predeterminado	Pre
1	timestamp	dpid	src_mac	dst_mac	src_ip	dst_ip	packet_count	byt
2	1745012768.32884	28792316550400	d6:a7:98:c7:f0:45	33:33:00:00:00:fb	N/A	N/A	1	218
3	1745012768.33089	28792316550400	6a:6c:d7:bf:17:12	33:33:00:00:00:fb	N/A	N/A	1	206
4	1745012768.34138	28792316550400	d6:a7:98:c7:f0:45	33:33:00:00:00:fb	N/A	N/A	2	107
5	1745012768.41708	28792316550400	f6:c7:e5:a7:3a:82	33:33:00:00:00:16	N/A	N/A	1	110
6	1745012768.45977	28792316550400	82:45:6e:54:3a:90	33:33:00:00:00:fb	N/A	N/A	1	218
7	1745012768.4731	28792316550400	f6:c7:e5:a7:3a:82	33:33:00:00:00:fb	N/A	N/A	2	206
8	1745012768.47919	28792316550400	6a:6c:d7:bf:17:12	33:33:00:00:00:16	N/A	N/A	2	90

Figura 21. Dataset de 500 entradas de tráfico.

	Predeterminado	Predeterminado	Predeterminado	Predeterminado	Predeterminado	Predetermina	Predeterr
1	timestamp	dpid	src_mac	dst_mac	src_ip	dst_ip	packet_c
2	1745027849.45357	28792316550400	56:7a:02:ca:b5:60	33:33:00:00:00:16	N/A	N/A	1
3	1745027849.4581	28792316550400	56:7a:02:ca:b5:60	33:33:00:00:00:02	N/A	N/A	2
4	1745027849.46384	28792316550400	56:7a:02:ca:b5:60	33:33:00:00:00:16	N/A	N/A	3
5	1745027849.51486	28792316550400	62:f7:3b:65:bc:e8	33:33:00:00:00:fb	N/A	N/A	1
6	1745027849.51852	15941488069632	62:f7:3b:65:bc:e8	33:33:00:00:00:fb	N/A	N/A	2
7	1745027849.54443	28792316550400	62:f7:3b:65:bc:e8	33:33:00:00:00:16	N/A	N/A	3
8	1745027849.54656	15941488069632	62:f7:3b:65:bc:e8	33:33:00:00:00:16	N/A	N/A	4

Figura 22. Dataset de 800 entradas de tráfico.

	Predeterminado	Predeterminado	Predeterminado	Predeterminado	Predeterminado	Predeterminad	Predet
141	1745034949.53044	6070364733184	02:2c:32:07:a9:b1	00:17:ee:bc:46:f1	N/A	N/A	5
142	1745034949.65279	5042306141184	02:67:bc:b9:48:54	00:17:ee:bc:46:f1	N/A	N/A	1
143	1745034949.65494	28792316550400	02:67:bc:b9:48:54	00:17:ee:bc:46:f1	N/A	N/A	2
144	1745034949.6573	33328084990720	02:67:bc:b9:48:54	00:17:ee:bc:46:f1	N/A	N/A	3
145	1745034949.65871	6070364733184	02:67:bc:b9:48:54	00:17:ee:bc:46:f1	N/A	N/A	4
146	1745034949.6594	15941488069632	02:67:bc:b9:48:54	00:17:ee:bc:46:f1	N/A	N/A	5
147	1745034949.78417	5042306141184	02:4a:31:c3:68:86	00:17:ee:bc:46:f1	N/A	N/A	1
148	1745034949.79143	28792316550400	02:4a:31:c3:68:86	00:17:ee:bc:46:f1	N/A	N/A	2

Figura 23. Dataset de 1100 entradas de tráfico.

Anexo 3. Pruebas de ataque

En esta sección se observa las diferentes pruebas de ataque echo con cada modelo entrenada al ser implementado en el entorno en conjunto con el modelo de seguridad.

Tabla 12. Pruebas de alertas con el modelo 1.

Pruebas	Ataques MAC Spoofing generados	Ataques identificados con el modelo 1	Precisión
1	30	19	0,67
2	40	27	0,65
3	60	39	0,66
4	80	53	0,68
5	90	59	0,67
6	100	65	0,65
7	150	99	0,64
8	200	134	0,69
9	250	165	0,67
10	300	201	0,66

Tabla 13. Pruebas de alertas con el modelo 2.

Pruebas	Ataques MAC Spoofing generados	Ataques identificados con el modelo 2	Precisión
1	30	24	0,83
2	40	32	0,84
3	60	50	0,85
4	80	67	0,82
5	90	76	0,84
6	100	84	0,86
7	150	126	0,83
8	200	168	0,85
9	250	210	0,84
10	300	252	0,83

Tabla 14. Pruebas de alertas con el modelo 3.

Pruebas	Ataques MAC Spoofing generados	Ataques identificados con el modelo 3	Precisión
1	30	26	0,91
2	40	37	0,92
3	60	55	0,93
4	80	73	0,92
5	90	82	0,91
6	100	92	0,94
7	150	138	0,92
8	200	183	0,93
9	250	230	0,91
10	300	276	0,92

Anexos 4. Resultados de los modelos entrenados

En esta sección se observa los parámetros obtenidos al entrenar con el algoritmo cada uno de los Dataset.

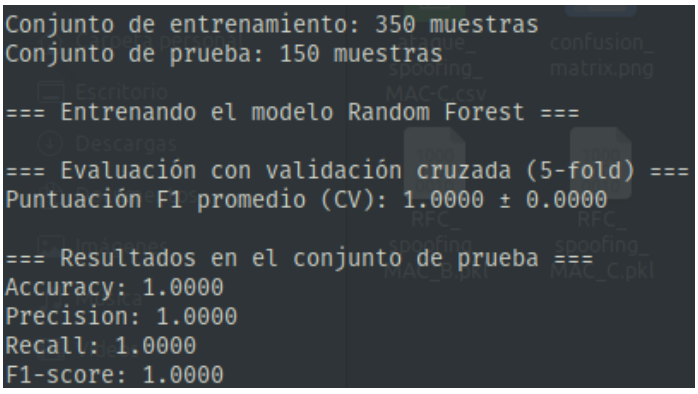


Figura 24. Resultados del entrenamiento del Dataset de 500 entradas

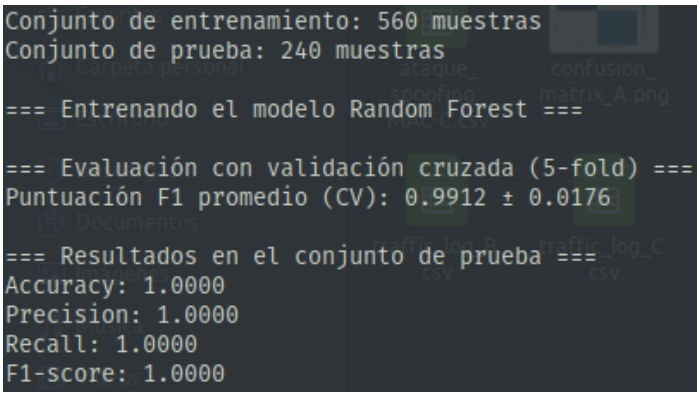


Figura 25. Resultados del entrenamiento del Dataset de 800 entradas

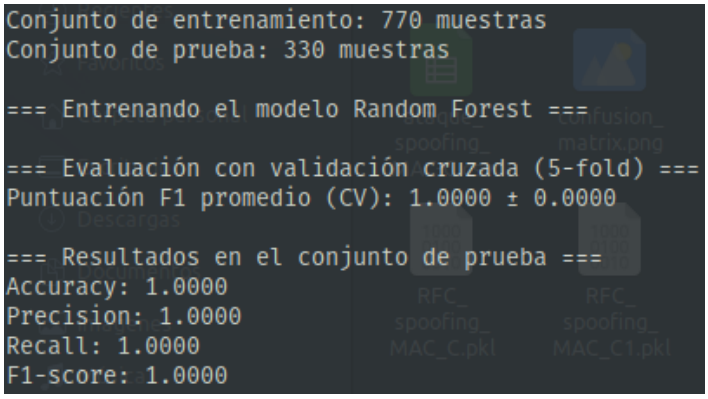


Figura 26. Resultados del entrenamiento del Dataset de 1100 entradas