UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE CUENCA

CARRERA DE TELECOMUNICACIONES

OBTENCIÓN DE UN MODELO DE DETECCIÓN DE ATAQUES DE DENEGACIÓN DE SERVICIOS EN REDES SDN EN MININET MEDIANTE APRENDIZAJE DE MÁQUINA

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

AUTOR: OSCAR IVAN TORRES AGUILAR

TUTOR: ING. JUAN DIEGO JARA SALTOS

Cuenca – Ecuador

2023

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Oscar Ivan Torres Aguilar con documento de identificación N° 0705322329; manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 22 de agosto del 2023

Atentamente,

Oscar Ivan Torres Aguilar
0705322329

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Oscar Ivan Torres Aguilar con documento de identificación N° 0705322329, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto Técnico: "Obtención de un modelo de detección de ataques de denegación de servicios en redes SDN en mininet mediante aprendizaje de máquina", el cual ha sido desarrollado para optar por el título de: Ingeniero en Telecomunicaciones, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 22 de agosto del 2023

Atentamente,

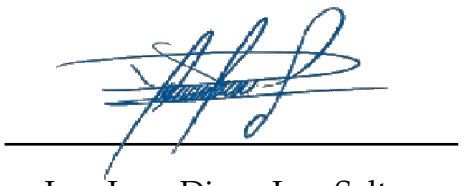
Oscar Ivan Torres Aguilar
0705322329

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Juan Diego Jara Saltos con documento de identificación N° 0103543658, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: OBTENCIÓN DE UN MODELO DE DETECCIÓN DE ATAQUES DE DENEGACIÓN DE SERVICIOS EN REDES SDN EN MININET MEDIANTE APRENDIZAJE DE MÁQUINA, realizado por Oscar Ivan Torres Aguilar con documento de identificación N° 0705322329, obteniendo como resultado final el trabajo de titulación bajo la opción proyecto técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 22 de agosto del 2023

Atentamente,



Ing. Juan Diego Jara Saltos
0103543658

AGRADECIMIENTOS

Agradezco en primer lugar a Dios, por guiarme con su luz y haberme dado la templanza para no ceder en mis anhelos por prepararme y por permitirme estar rodeado de las personas correctas, de quienes mucho he aprendido tanto, a mi esposa por su apoyo incondicional, las actos de amor que cada día me demuestra y por ser mi motivación para seguir esforzándome, a mi madre por su amor sin limites y por demostrarme con su fortaleza y su fe a no desistir, a mis suegros por tener fe en mi y apoyarme en los momentos mas difíciles y por darme ejemplo de que el apoyo de esposos es una de las mas grandes fortalezas, a mis hermanos y cuñados por su preocupación a lo largo de este proceso y a mi tutor, ingeniero Juan Diego Jara, por tomarse el tiempo para guiarme en el desarrollo de este trabajo.

DEDICATORIA

DEDICATORIA

Este trabajo va dedicado a Dios, a mi esposa y al mas hermoso obsequio que lleva en su vientre, mi hijo, a mi familia, a mis suegros y mis cuñados por creer en mi.

Índice general

Índice de figuras

Índice de tablas

Resumen

Al paso de los últimos años la alta demanda de conectividad ha hecho que el internet crezca a pasos agigantados a nivel global, lo que ha impulsado la generación de nuevas tecnologías que permitan la escalabilidad de las redes y con ello también el desarrollo de otros aspectos inherentes, como la seguridad. Las redes definidas por software (software defined network -SDN-) son una propuesta que ha llamado mucho la atención debido a su versatilidad al tener un control centralizado, pero pese a las ventajas que en cuanto a flexibilidad y escalabilidad que tiene, aún existen vacíos de seguridad en su estructura que deben ser resueltos para poder ser desplegadas ampliamente. En el presente documento se analizan algunas características de la arquitectura de las redes definidas por software que presentan vulnerabilidades de seguridad y los ataques más comunes que pueden sufrir, prestando principal interés en los ataques de denegación de servicios de la capa de control, específicamente en aquellos ataques que saturan al controlador, que es el elemento central o cerebro de las SDN y se presenta un modelo que permita predecir dichos ataques mediante un algoritmo del tipo árbol de clasificación. La generación del dataset para el entrenamiento se genera a partir de una topología de red diseñada y simulada en el software mininet y complementada con un dataset externo; el controlador evaluado es RYU, que es, además, el marco de desarrollo en donde se implementa el modelo para la detección de ataques. Se muestran las evaluaciones sobre la precisión y eficiencia del modelo generado frente a posibles ataques y a modificaciones del escenario propuesto y finalmente se presentan conclusiones y trabajos futuros.

Palabras clave: SDN; mininet;ciberseguridad;aprendizaje automatico;RYU

Abstract

In recent years, the high demand for connectivity has caused the Internet to grow by leaps and bounds globally, which has driven the generation of new technologies that allow the scalability of networks and also the development of other inherent aspects, such as security. Software-defined networks (SDN) are a proposal that has attracted much attention due to their versatility in having centralized control, but despite the advantages in terms of flexibility and scalability, there are still security gaps in their structure that must be resolved in order to be widely deployed. This paper analyzes some characteristics of the architecture of software-defined networks that present security vulnerabilities and the most common attacks that they can suffer, paying special attention to denial-of-service attacks in the control layer, specifically those attacks that saturate the controller, which is the central element or brain of SDNs, and presents a model that allows predicting such attacks using a classification tree type algorithm. The generation of dataset for training is generated from a network topology designed and simulated in the mininet software and complemented with an external dataset; the evaluated controller is RYU, which is also the development framework where the model for attack detection is implemented. Evaluations on the accuracy and efficiency of the generated model against possible attacks and modifications of the proposed scenario are shown and finally, conclusions and future work are presented.

Keywords: SDN; mininet;cybersecurity;machine learning;RYU

Antecedentes

Si bien las redes definidas por software (SDN) son una novedosa tecnología que permite solventar problemas presentes en la gestión de las redes tradicionales rígidas, también trae consigo nuevos desafíos, en especial en la seguridad; esta ha sido la piedra en el camino que ha impedido que estas redes se desplieguen rápidamente a diferencia de las redes tradicionales.

La seguridad de las redes es imprescindible, tanto a nivel corporativo como personal, ya que gran parte del tiempo se trata con información sensible, por ello la integridad de estos datos debe ser garantizada.

Los ataques de denegación de servicios (DoS) de diferente tipo son los ataques más comunes que sufren las redes tradicionales y las SDN no son la excepción. Este tipo de ataques generan problemas en el funcionamiento de los servicios y en el caso de las redes definidas por software, al centralizar el control de la red, también puede ser víctima de un ataque de esta índole. Estos ataques no suelen suceder aislados, por lo general son precedentes para otro tipo de ataques como secuestro de datos o ramsonware.

En los últimos años, las técnicas para ataques por denegación de servicios han evolucionado, actualmente es común que estos ataques se realicen desde diferentes dispositivos, algo que se conoce como ataque de denegación de servicio distribuido o (DDoS). En 2019 el colectivo Anonymous lanzó ataques DDoS al banco central del Ecuador como represalia ante el gobierno por su decisión de retirar el asilo a Julian Assange, este ataque hizo que el sistema del banco permaneciera fuera de línea, se estima que se realizaron cerca de 40 millones de ataques. En el 2021 el banco del Pichincha sufrió un ataque en su sistema, que inhabilitó gran parte de sus servicios. Todos estos ataques se dieron en las redes tradicionales, en las que actualmente se

trabaja arduamente para crear nuevas técnicas que permitan prevenir y solventar problemas de ciberseguridad, sin embargo las redes definidas por software tienen la ventaja de que la implementación de los sistemas o algoritmos de seguridad son más fáciles de implementar debido a su control centralizado, que permite una rápida y mejor gestión.

Justificación

Entre 2019 y 2020, debido a la pandemia COVID-19, el uso de internet tuvo un crecimiento brusco debido a la virtualización de muchos servicios y a la transaccionalidad de las diferentes entidades financieras, aunado al crecimiento que han venido teniendo plataformas de servicio como por ejemplo plataformas de streaming tales como Spotify, Netflix, Prime video, etc. La arquitectura de las redes de internet no fue originalmente diseñada para dar soporte a tanta demanda, un ejemplo de este hecho es el agotamiento de las direcciones IPv4, por lo que se ha creado la actualización IPv6; pero la actualización de protocolos, arquitectura, sistemas de seguridad, entre otros, no es una tarea fácil en las redes tradicionales debido a su rigidez, y las redes definidas por software (SDN) son la evolución tecnológica que se necesitaba para ventilar muchos de estos dilemas.

Las SDN, no obstante, al ser una tecnología relativamente nueva, aún están sujetas a vulnerabilidades que necesitan ser resueltas antes de ser empleadas de forma masiva. Los ciberdelitos son más frecuentes hoy en día y es necesario evaluar las debilidades y crear soluciones que le den fiabilidad a esta nueva tecnología.

Gracias a la programabilidad que se tiene, las redes definidas por software permiten crear algoritmos fáciles de implementar y gestionar en la red, para paliar estos problemas de seguridad. En este proyecto se presenta un sistema para la detección de ataques por denegación de servicios, obtenido mediante la técnica de aprendizaje supervisado en una red SDN para evitar posibles ataques que perjudiquen al correcto funcionamiento de la red.

Objetivos

Objetivo General

- Obtener un modelo de detección de ataques por denegación de servicios a redes definidas por software en un entorno simulado en Mininet mediante aprendizaje de máquina.

Objetivos específicos:

- Revisar el estado del arte de las herramientas de software para SDN.
- Implementar un entorno de red definida por software en el emulador Mininet.
- Recolectar datos (Dataset) del estado de la red frente a ataques DoS.
- Entrenar el modelo de aprendizaje de máquina utilizando el dataset y aplicarlo al controlador de la SDN.
- Validar el modelo

Introducción

La seguridad e integridad de la información que transmitimos y recibimos constantemente mediante medios de telecomunicaciones se ve amenazada constantemente. En la actualidad esa información es sensible y de vital importancia en muchos casos, como cuando se realizan transferencias bancarias con sumas altas de dinero, cuando enviamos información personal al crear cuentas en entidades de diferente índole, cuando enviamos información personal o simplemente cuando se realiza cualquier trabajo diario. En cualquiera de los casos existen situaciones en donde personas con malas intenciones pueden afectarnos, ya se con el fin de conseguir algo de nosotros o solo por el hecho de hacernos daño, es por eso que a la par con el crecimiento y evolución de las redes de telecomunicaciones, también se han desarrollado diversas técnicas para mitigar estas amenazas.

Los ciberataques son mucho mas frecuentes hoy en día que hace un par de decadas, y es entendible por el hecho de la gran expansión que ha tenido el internet alrededor de todo el mundo. Muchos ataques se realizan con fines económicos, es decir, para robar dinero de cuentas bancarias o para robar información importante para la víctima y luego pedir dinero a cambio de devolver tal información, este tipo de ataque se conoce como Ramsomware; estos tipos de ataques se realizan únicamente con el propósito de dañar a la víctima, como sucede cuando se realizan ataques por denegación de servicios, en donde se busca saturar los recursos computacionales de los equipos de la víctima con el fin de que estos queden imposibilitados para brindar servicios de forma normal.

Sea cual fuere la situación, los ciberataques son un problema en un mundo globalizado y que ha hecho del internet la herramienta mas importante para las comunicaciones y por este motivo es imperativo crear herramientas que protejan a

los usuarios de estos ataques.

Las redes de internet, tal y como las conocemos hoy en día, también son conocidas como redes tradicionales, para diferenciarlas de nuevas tecnologías que han estado surgiendo en los últimos años. Se han creado una gran cantidad de soluciones para prevenir ataques en este tipo de redes sin embargo, tienen el problema de ser rígidas, es decir, no son fácilmente administrables, por lo que modificaciones en la red como podrían ser actualizaciones de protocolos, balanceo de carga, actualización de firmware, etc, se vuelve una tarea compleja y tediosa en redes de gran tamaño. La redes definidas por software son la tecnología emergente que busca solucionar estos problemas al centralizar su control, sin embargo esta nueva tecnología aun requiere de estudios y mejoras para que pueda ser confiable, ya que así como trae consigo nuevas propuestas, también acarrea nuevos desafíos sobre todo en cuanto a seguridad y una de estas debilidades se debe al hecho de que las SDN separan el plano de datos del plano de control [1].

En este documento se evalúa las vulnerabilidades en cuanto a seguridad que tienes esta nueva arquitectura de red y se presenta el desarrollo de un modelo que permite prevenir un tipo de ataque de denegación de servicios, que precisamente trabaja con una clase de paquetes llamados 'PACKET_IN' exclusivos de las SDN.

Capítulo 1

Marco teórico

En este capítulo se presentan conceptos, definiciones y algunos aspectos teóricos sobre las redes SDN, seguridad, algoritmos de inteligencia artificial, etc. relacionados al desarrollo de este trabajo

1.1. Redes definidas por software (SDN)

1.1.1. Arquitectura de las SDN

Las Redes Definidas por Software (Software Defined Network o simplememte SDN) constituyen una nueva manera de afrontar la administración de redes. A diferencia de las redes tradicionales, donde cada dispositivo, como routers o switches, tienen su propio sistema de control del tráfico de red, las SDN dividen el trabajo entre dos componentes o planos principales que son el plano de control y el plano de datos [2].

En primer lugar, se tiene el plano de control, que es el responsable de tomar decisiones estratégicas sobre la dirección del tráfico de red. Este componente es centralizado, en general, en lo que se conoce como un controlador de red SDN. Es el encargado de gestionar los componentes de red de nivel superior e inferior[1].El objetivo es dar a los administradores de la red un punto de control único que permita tener una visión y gestión unificada de toda la red.

Por otro lado, está el plano de datos, que ejecuta las acciones de manejo

del tráfico con base en las decisiones tomadas por el plano de control. Estos dos componentes se comunican entre sí mediante protocolos, como OpenFlow. Consiste principalmente en Elementos de Reenvío (FEs) incluyendo switches tanto físicos como virtuales que pueden ser accedidos por medio de una interfaz abierta, permitiendo la conmutación y reenvío de paquetes[3].

La ventaja de las SDN radica en su flexibilidad y capacidad de simplificar la gestión de la red. Estos sistemas permiten implementar y actualizar políticas de seguridad de red de manera más efectiva, manejar la virtualización de redes y realizar una gestión avanzada de las redes. Aunque las SDN están en constante desarrollo y su implementación puede variar entre diferentes organizaciones, se están convirtiendo en una parte fundamental de la administración de redes moderna. La Figura 1.1 muestra un esquema sencillo de la arquitectura típica de una red definida por software.

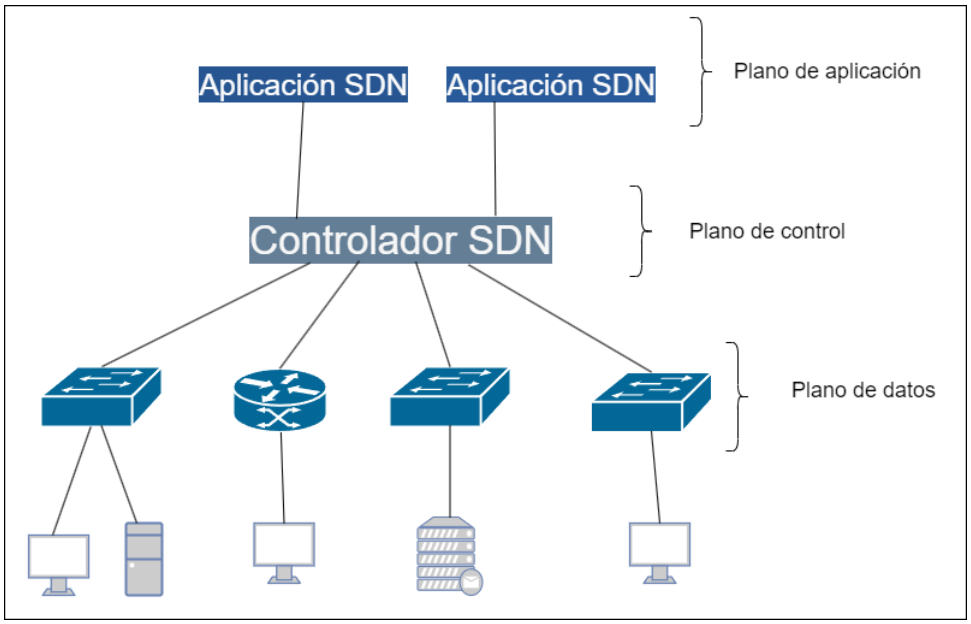


Figura 1.1: Arquitectura simplificada de una SDN.

La SDN tiene un gran potencial para revolucionar la manera en la que operan las redes, y se ha promocionado en particular a OpenFlow como una " idea revolucionaria en la red". Los beneficios propuestos abarcan desde algoritmos simplificados, control centralizado, la comercialización del hardware de red, la eliminación de los intermediarios, hasta la posibilidad de diseñar e implementar

'aplicaciones' de terceros [4].

Las interfaces que permiten la comunicación entre las capas de control y aplicación se la conoce como *interfaz Northbound*, esto es, una interfaz común para el desarrollo de aplicaciones. Típicamente, una interfaz northbound realiza una abstracción de instrucciones de bajo nivel que son utilizados por las interfaces hacia el sur (southbound) para programar dispositivos de reenvío. Por lo general se trata de APIs que son conjuntos de reglas y protocolos que permiten que las capas se comuniquen entre si La API de la *interfaz southbound*, por otro lado, permite establecer la comunicación entre la capa de datos y la capa de control y es la que establece el conjunto de instrucciones que usan los dispositivos de reenvío. Además, esta interfaz también determina cómo se comunican estos dispositivos con los componentes del plano de control. Este protocolo establece específicamente cómo se lleva a cabo la interacción entre los componentes del plano de datos y del plano de control[5]. La figura 1.2 muestra gráficamente la ubicación de las interfaces en la arquitectura de una red SDN, por ejemplo la API Northbound se conecta con la GUI del administrador en la capa de aplicación mediante la API RESTful.

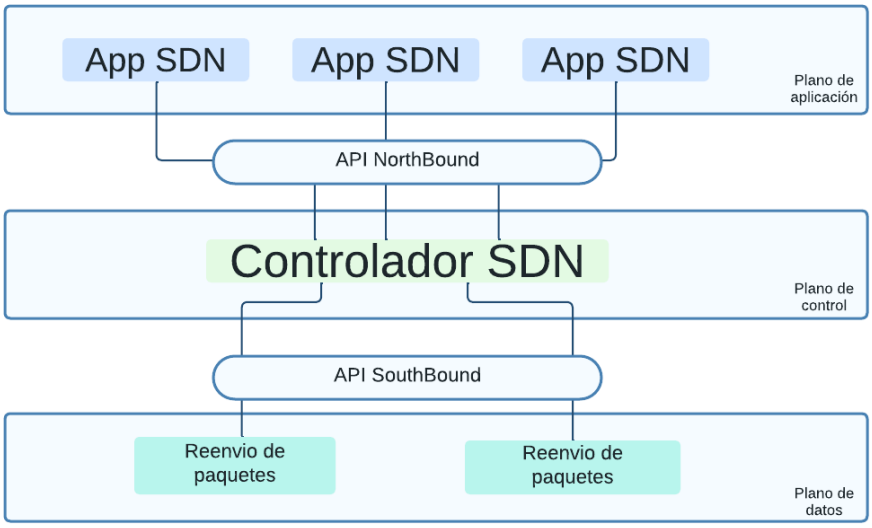


Figura 1.2: Interfaces Northbound y Southbound que permiten la comunicación entre capas de la arquitectura de SDN.

1.1.2. Protocolos de comunicación

Existen diversos protocolos de comunicaciones en las SDN, que son los que permite a los controladores de red dirigir el tráfico en los switches de red.

Algunas de las principales funciones de los protocolos de comunicaciones son:

- **Comunicación entre capas:** Los protocolos de comunicación en SDN permiten la interacción entre el plano de control (en donde reside la lógica de red) y el plano de datos (donde se encuentran los dispositivos de reenvío, como los switches). Un protocolo de este tipo, como OpenFlow, habilita al plano de control para que manipule el comportamiento de reenvío de paquetes de los dispositivos que se encuentran en el plano de datos.

- **Programación de dispositivos de red:** Los protocolos permiten la programación de las tablas de flujo de los nodos de la red, lo que permite definir cómo se deben manejar y enrutar los paquetes de red.

- **Abstracción de la infraestructura de red:** Los protocolos de SDN pueden abstraer los detalles de la infraestructura de red, lo que permite a los operadores y aplicaciones de red interactuar con la ella a un nivel más alto de abstracción.

- **Interoperabilidad:** Al ser protocolos estándar, permiten la interoperabilidad entre diferentes dispositivos y fabricantes de red, lo que facilita la flexibilidad y elección en el despliegue de la red.

- **Gestión centralizada:** Los protocolos de comunicación de SDN permiten la gestión centralizada de la red, lo que puede facilitar la administración de la red y la instauración de coherentes políticas de red.

Entre los protocolos mas conocidos para las SDN estan: **Border Gateway Protocol (BGP)** que es un protocolo que se utiliza para el intercambio de información de enrutamiento y para la toma decisiones de cómo enrutar el tráfico de datos entre diferentes redes autónomas (AS, por sus siglas en inglés). Mediante BGP, los sistemas autónomos tienen la capacidad de establecer conexiones tanto internas como con otros sistemas, permitiendo un intercambio eficiente de paquetes de datos. Los

componentes esenciales de BGP, incluyendo su versión más reciente, BGPv4, están exhaustivamente descritos en el RFC 1163. [6].

NETCONF es un protocolo para la administración de redes, desarrollado por el Grupo de Tareas de Ingeniería de Internet (IETF). Ofrece a los profesionales de la red, un medio seguro para configurar los nodos de la red como firewalls, switches, routers y otros.

Protocolo de Presencia y Mensajería Extensible (XMPP) Pprotocolo basado en el lenguaje de marcado extensible XML. Se utiliza en mensajería instantánea y para la identificación de presencia en línea. El protocolo funciona en servidores y es casi en tiempo real [7].

El **Protocolo de Gestión de Base de Datos Open vSwitch (OVSDB)** es un protocolo de configuración asociado al protocolo OpenFlow, cuyo objetivo es la administración de las instancias de Open vSwitch que es un conmutador de red virtual de código abierto que ha sido diseñado para utilizarse en entornos de red virtualizados, como infraestructuras de nube o en contenedores[8].

El **Perfil de Transporte MPLS (MPLS-TP)** es una variante de la tecnología MPLS pensada para usarse como una capa de red en infraestructuras de transporte. Las ampliaciones a MPLS para cumplir con estas funciones están siendo desarrolladas por la IETF, basándose en las necesidades expresadas por los proveedores de servicios[9].

1.1.3. OpenFlow

Uno de los protocolos extensamente utilizados es **OpenFlow**; este es un protocolo de código abierto que se originó en la Universidad de Stanford después del Proyecto Clean Slate1. Este protocolo se propuso para permitir a los investigadores probar nuevos protocolos en experimentos en las redes del campus que utilizaban a diario. Actualmente, la ONF (Open Networking Foundation), es la encargada de impulsar activamente los avances de las SDN y la estandarización de OpenFlow, muchos controladores y equipos de red utilizan este protocolo como su protocolo de comunicación

Algunos de los principales aspectos de OpenFlow se describen a continuación:

- **Abierto y Estándar:** OpenFlow es un estándar abierto y ampliamente aceptado en la industria de las redes. Esto significa que múltiples fabricantes pueden implementar el protocolo en su hardware de red, lo que permite la interoperabilidad y la elección de hardware.

- **Separación de plano de control y plano de datos:** Una característica fundamental de SDN es la disociación de los planos de datos y control. OpenFlow se utiliza para comunicar las instrucciones de control desde el controlador SDN central hacia los nodos de la red, permitiendo que la lógica de control se ubique de forma centralizada en lugar de estar distribuida en cada dispositivo de red.

- **Flujo de Reglas:** Los dispositivos de las redes tradicionales tomaban decisiones de enrutamiento y reenvío basados en protocolos de enrutamiento tradicionales, en SDN los dispositivos OpenFlow operan mediante "flujos de reglas"que se programan desde el controlador SDN. Cada flujo de reglas especifica cómo se debe manejar un tipo particular de tráfico, como redirigirlo a un puerto específico o aplicar una política de seguridad.

- **Gestión Dinámica de Redes:** OpenFlow permite una gestión de red más dinámica y flexible. Los administradores de red pueden adaptar la configuración de los equipos de la red de acuerdo a las necesidades y en tiempo real , sin necesidad de cambios manuales en la configuración de cada dispositivo.

- **Innovación y Experimentación:** OpenFlow ha fomentado la innovación en la gestión y el control de redes al proporcionar una plataforma programable y abierta. También ha sido utilizado en entornos de investigación y educación para experimentar con nuevas ideas en el campo de las redes.

1.1.4. Controladores

Un controlador SDN se puede conceptualizar como el cerebro de la red. En esta arquitectura, el controlador es la entidad que aloja el plano de control, diferenciado del plano de datos, que se limita a enviar los paquetes a su destino.

Algunas de las principales funcionalidades son:

- Responsabilidad en la toma de decisiones: En la arquitectura de SDN, el controlador es quien dicta cómo se deben encaminar los paquetes en la red. Si un switch encuentra un paquete cuyo tratamiento desconoce, acude al controlador SDN para que decida cómo manejarlo.
- Panorámica de la red: Gracias a que todas las decisiones de encaminamiento fluyen a través del controlador SDN, este posee una visión panorámica y completa de la red. Esto permite un encaminamiento más efectivo y la posibilidad de aplicar políticas de red a nivel global.
- Punto de acceso para la programación de la red: El controlador SDN brinda un punto de acceso que los administradores de red pueden utilizar para programar y gestionar la red. Por medio de este acceso, los administradores pueden establecer políticas de red, administrar el tráfico y configurar los dispositivos en la red.
- Comunicación con los dispositivos de red: El controlador SDN dialoga con los nodos de red, como los switches y routers, mediante el protocolo OpenFlow u otros protocolos de SDN. Este controlador puede dar instrucciones a los dispositivos sobre cómo deben tratar los paquetes de datos.

Existen distintos tipos de controladores SDN, que pueden ser de código abierto o proporcionados por proveedores específicos, y que pueden variar en cuanto a su arquitectura y características. Sin embargo, todos cumplen la función esencial de tomar decisiones de encaminamiento y ofrecer un punto de acceso para la programación de la red en un entorno SDN.

Los controladores pueden ser propietarios o de código abierto que son accesibles para el publico de manera gratuita. Algunos de los mas utilizados son:

- **OpenDaylight (ODL):** Este controlador SDN es uno de los más populares y es código abierto. Está desarrollado y respaldado por la Fundación Linux. OpenDaylight ofrece una plataforma de desarrollo extensible y modular que permite a los desarrolladores añadir funciones y protocolos según sea necesario.

- **ONOS (Open Network Operating System):** ONOS es otro controlador de código abierto popular. Fue diseñado para suplir los requerimientos de los operadores de redes y cuenta con un rendimiento, escalabilidad, disponibilidad y abstracciones de red superiores para simplificar la creación de aplicaciones y servicios de red.

- **Floodlight:** Floodlight es un controlador OpenFlow escrito en Java. Es bastante popular debido a su facilidad de uso y su interfaz de programación de aplicaciones (API) bien diseñada. Floodlight puede trabajar con cualquier dispositivo de red que sea compatible con OpenFlow. También es de código abierto.

- **Ryu:** Ryu es un controlador SDN de código abierto desarrollado en Python. Su enfoque modular y su flexibilidad hacen que sea una opción popular para la investigación académica y para aquellos que desean experimentar con el desarrollo de SDN. Este es el controlador con el que trabajó en este proyecto; en la siguiente sección se detallan mas características sobre el mismo

- **Cisco APIC-EM:** Este es un controlador SDN desarrollado por Cisco. APIC-EM es parte de la arquitectura de red basada en intenciones de Cisco y proporciona funciones de automatización y orquestación de red.

- **Juniper Contrail:** Contrail es de código abierto y es un controlador SDN desarrollado por Juniper Networks. Proporciona funciones de red definida por software para la nube privada, la nube pública y las implementaciones de nube híbrida.

- **HP VAN (Virtual Application Networks) SDN Controller:** Desarrollado por HP Enterprise, este controlador es una solución comercial que proporciona funcionalidades de SDN y se integra con la suite de aplicaciones de red de HP.

Controlador RYU

La palabra Ryu en japones significa dragón. Ryu es un controlador para SDN desarrollado en los laboratorios Nippon Telegraph and Telephone Corporation (NTT),

Software Innovation Center (SIC) en Japón y lanzado por primera vez en 2012. Fue lanzado bajo la licencia de código abierto Apache 2.0.

Este controlador proporciona bibliotecas y herramientas para ensamblar redes SDN de manera cómoda y sencilla y es compatible con varias versiones de OpenFlow 1.0, 1.2, 1.3, 1.4, 1,5. Se considera un software basado en componentes de código abierto y definido por un marco de trabajo en red e implementado completamente en Python. Otro punto fuerte de Ryu es que admite múltiples protocolos orientados southbound (protocolos que permiten la comunicación entre los dispositivos de la capa de control y de la capa de datos) para administrar dispositivos, como OpenFlow, OpenFlow Management and Configuration Protocol (OF-Config), Network Configuration Protocol (NETCONF), y otros. También es compatible con las extensiones de Nicira[10].

Ryu provee el entorno en donde se puede administrar, manejar y monitorear el funcionamiento de la red. Tambien posee una serie de aplicaciones de ejemplo en python, una de estas es 'simple_switch_13', que es una aplicacion que permite simular el comportamiento de un switch openflow, esto ayuda en la simulación de redes.El logo de este controlador es un caracteristico dragon, como se observa en la siguiente figura.

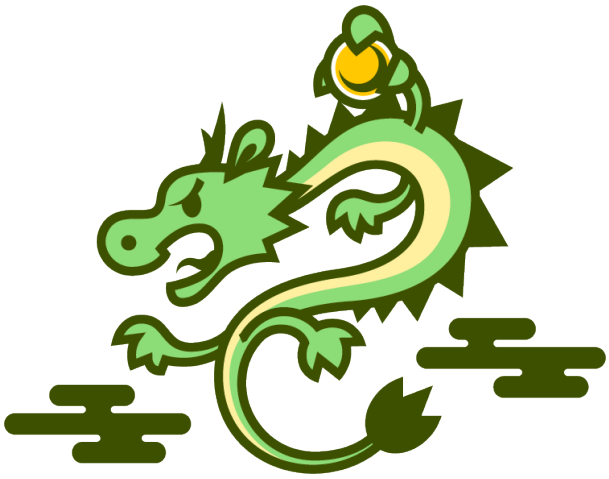


Figura 1.3: Logo del controlador RYU

1.2. Mininet

Mininet es un simulador de redes y de código abierto centrado en el protocolo OpenFlow y las SDN, que se utiliza para fines de desarrollo, formación e investigación. Es capaz de funcionar en máquinas de computación básicas o con recursos limitados de procesamiento. Mininet proporciona un entorno de línea de comandos fácil de usar que facilita la interacción entre la red virtual y el usuario o administrador de red . Ofrece también una API en Python para la creación y gestión de redes de datos.[11].

Es uno de los primeros simuladores que se desarrollaron específicamente para la creación de SDN y que permite la creación y ejecución de redes desde pequeña escala con ancho de banda y tráfico artificial en equipos con bajos recursos computacionales, su licencia es libre (BSD –Berkely Software Distribution). Sin embargo, las simulaciones están limitadas a las capacidades del host anfitrión, es decir, una red copleja con una cantidad considerable de equipos como hosts, switches, controladores, etc. consume una mayor cantidad de procesamiento y memoria. Mininet es versátil y permite el uso de controladores que se ejecutan en maquinas remotas, esto es de gran utilidad ya que permite que la topología de red tenga un comportamiento mas realista[12].

Mininet posee una aplicación de ejemplo desarrollada en python llamada *miniedit* que es una herramienta de interfaz gráfica de usuario (GUI) para diseñar y probar redes. Miniedit provee de una herramienta sencilla para crear topologías de red simplemente arrastrando componentes e interconectándolos, la figura 1.4 muestra la interfaz de miniedit.

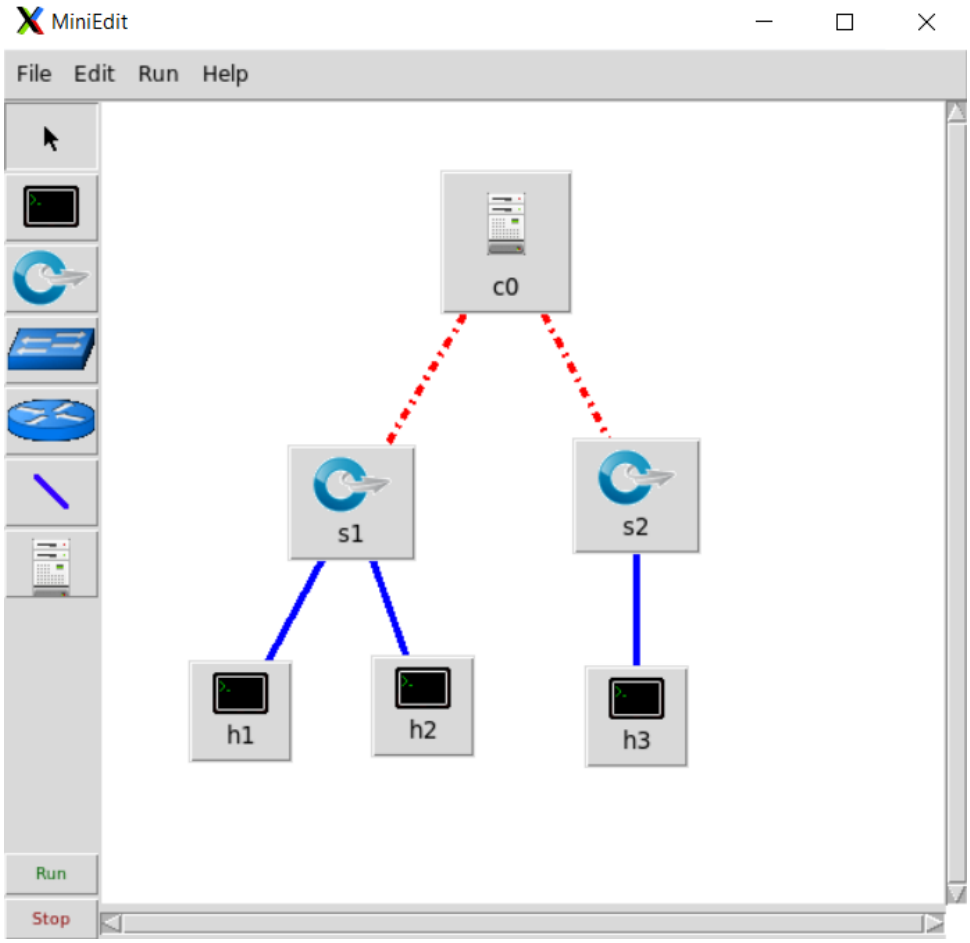


Figura 1.4: Interfaz gráfica de miniedit con una pequeña topología.

En la figura 1.4 se puede observar un controlador en la parte superior, dos switches nombrados *s1* y *s2* y 3 hosts ubicados en la parte inferior.

Mininet permite crear topologías de diferentes formas, una es directamente al ejecutar mininet pasándole algunos parámetros o simplemente ejecutando "**sudo mn**" que crea una topología con un controlador conectado a un switch y este conectado a dos hosts[13], otra forma es mediante un archivo en python en donde se debe importar las librerías de mininet y otra manera es mediante miniedit, cada una tiene sus ventajas, dependiendo de lo que se desee realizar. Si bien realizar topologías de forma gráfica resulta mas sencillo que insertarlas por código, también tiene sus limitaciones, ya que algunos parámetros de la red no son configurables o no de manera sencilla. Cuando se ejecuta miniedit, se puede indicar que se abra la consola de mininet cuando se ejecuta la topología y mediante esta consola se pueden modificar algunos valores, pero esto resulta engorroso si se trabaja con una topología grande. Crear

topologías en python es una de las mejores opciones si se quiere personalizar con mayor profundidad la red, por ejemplo, si deseamos indicar el ancho de banda en el enlace entre dos nodos, latencias, recursos computacionales de los nodos, etc. Pero si se desea crear una topología simple, que sigue un patrón, esta se puede crear de forma directa desde la linea de comandos, de la siguiente manera:

```
sudo mn --[op]=[param],[argumentos] --[op n]=[param n],[argumentos]
```

Algunos parámetros típicos son el tipo de topología, controlador a usar, cantidad de hosts y switches, etc. Los tipos de topologías pueden ser:

- Topología Mínima (Minimal Topology): Esta es la topología más simple y se crea por defecto cuando se inicia Mininet sin especificar una topología. Consta de un único switch que se conecta a dos hosts.
- Topología Simple (Single topology): La topología 'single' genera una red con un solo switch y un número determinado de hosts.
- Topología Invertida (Reversed topology): La topología 'reversed' genera una red con un solo switch y dos hosts, similar a la topología 'single', pero los nombres de los hosts y del switch se invierten.
- Topología de Línea (Linear Topology): Esta topología consta de una línea de interruptores, cada uno de los cuales está conectado a un host. El número de interruptores (y por lo tanto de hosts) es especificado por el usuario.
- Topología de Árbol (Tree Topology): Esta topología tiene un interruptor raíz que está conectado a varios interruptores hijo, cada uno de los cuales puede estar conectado a otros interruptores y a hosts. El número de niveles en el árbol y el factor de ramificación son especificados por el usuario.
- Topología de Torus (Torus Topology): Esta es una extensión de la topología de rejilla en la que los bordes de la rejilla también están conectados, formando un toro.

Se muestra un ejemplo a continuación de una topología lineal con dos switches y un host conectado a cada switch.

```
sudo mn --topo linear,2 --mac --switch ovsk
```

El parámetro *–topo linear,2* indica que la topología es lineal y que se crearán dos nodos en la red, *–mac* indica que cada host debe tener una dirección mac, *–switch ovsk* le indica que los switches deben ser del tipo ovsk (Open vSwitch Kernel Switch) que son un tipo de switch virtual utilizado en Mininet. Se tienen diferentes opciones al momento de elegir un switch, estos son algunos de los que provee mininet[11]:

ivs: IVS (Indigo Virtual Switch) es un conmutador virtual OpenFlow basado en el Indigo Core, un subconjunto de la pila de conmutadores Indigo que implementa la capa de abstracción de hardware de conmutador y OpenFlow.

ovs: OVS (Open vSwitch) es un switch virtual multicanal de producción que admite estándares de red estándar como VLANs, STP (Spanning Tree Protocol), IPsec, etc., así como OpenFlow.

ovsbr: Esto también se refiere a Open vSwitch, pero en este caso el conmutador se ejecuta en modo bridge, que es un dispositivo que conecta dos o más segmentos de red.

ovsk: hace referencia a Open vSwitch con soporte para el protocolo OpenFlow.

ovsl: Open vSwitch en el modo "solo aprendizaje"(learning switch mode), que es un tipo de switch que aprende los patrones de tráfico en su red y puede generar y modificar sus propias tablas de conmutación.

user: Conmutador de usuario de Mininet, que se ejecuta completamente en el espacio de usuario como un proceso normal. No tiene las capacidades de rendimiento del conmutador del kernel (como OVS), pero es más fácil de depurar porque se puede ejecutar con un depurador de software.

La API de python para Mininet, es una interfaz para la programación de aplicaciones y topologías de red, proporciona las bibliotecas del emulador para escribir código en Python que describa redes personalizadas por los usuarios. Los módulos generados con la API de Mininet se pueden interpretar ejecutando scripts de

Python o utilizando la consola del emulador Mininet en donde se ejecuta el comando y el parámetro "sudo mn –custom=<file.py>". Con esta API, es posible abstraer redes utilizando la mayoría de las clases u objetos presentes en el emulador Mininet[14].

Cuando se crea una topología mediante código en python se deben importar las bibliotecas necesarias de mininet, para que el interprete pueda ejecutar dicho código. algunas mas importantes son: **mininet** para importar la clases mininet desde el modulo 'mininet.net', **OVSKernelSwitch, Controller, RemoteController** esto permite importar el tipo de switch a utilizar y las librerias para el controlador, que puede ser remoto, es decir, que se ejecuta en algún servidor en una red diferente, **CLI** es el módulo que contiene la interfaz de línea de comandos de Mininet, que puedes usar para interactuar con tu red de Mininet después de que se ha iniciado, El módulo **util** es una colección de funciones que proporciona varias características útiles y conveniencias para trabajar con redes como ejecutar comandos en el sistema anfitrión, crear y manipular listas de nodos y enlaces, trabajar con direcciones IP y MAC, entre otros.

Para importar las librerías se lo realiza de la siguiente manera:

```
from mininet.net import Mininet
from mininet.node import OVSKernelSwitch,Controller,RemoteController
```

1.3. Docker

Docker es una plataforma para la creación y distribución de aplicaciones y es un entorno que permite ejecutar software de manera eficiente, encapsulándolas junto con todas sus dependencias en unidades portátiles llamadas " contenedores". Estos contenedores proporcionan un entorno aislado y coherente en el que las aplicaciones pueden funcionar de manera consistente en diversos entornos, desde máquinas locales hasta servidores en la nube, independientemente de las diferencias en la infraestructura subyacente y el sistema operativo. En resumen, Docker facilita el de desarrollo, implementación y gestión de aplicaciones al empaquetarlas junto con sus

componentes necesarios en contenedores autocontenidos y listos para ejecutarse.

1.4. IPERF

IPERF es una utilidad empleada para la medición y análisis del desempeño de redes mediante la generación de flujo de información entre dos puntos dentro de una red. Esta herramienta permite determinar la velocidad de transferencia de datos, la capacidad del ancho de banda y otros parámetros clave relacionados con la eficiencia de la red. iPerf se emplea para probar y optimizar el rendimiento de las redes al simular flujos de datos y analizar cómo responden en términos de velocidad y calidad de conexión. En esencia, iPerf es una utilidad valiosa para evaluar el funcionamiento y la capacidad de una red al generar tráfico controlado y medir su rendimiento en condiciones específicas. Además permite enviar trafico tipo TCP o UDCP especificando el tamaño de la trama, cantidad de bytes, tiempo de envío, etc. Esta última característica lo hace muy útil para simular tráfico en las redes.

1.5. Seguridad en redes SDN

Las redes tradicionales aun tienen problemas de seguridad que han tratado de resolverse permanentemente, sin embargo dichas soluciones solo presentan un alivio temporal debido a las actualizaciones tecnológicas que soportan los equipos de red y por lo tanto surgen nuevas vulnerabilidades que son aprovechadas por los atacantes. No todas las subredes que conforman la enorme red de internet son seguras y algunas están totalmente desactualizadas debido al gran costo que puede significar una actualización o simplemente porque la tecnología utilizada en estas redes no tiene mas soporte; cabe aquí resaltar nuevamente las ventajas de las SDN al ser una arquitectura que proporciona un marco que facilita la administración de las redes, sin embargo SDN tiene sus propios problemas con respecto a seguridad.

Cuando se trata sobre la seguridad en las SDN se hace referencia a las políticas y prácticas empleadas para protegerlas de ataques, intrusiones y fallas del sistema. Anteriormente se ha mencionado que las SDN se caracterizan por ser

altamente programables y gestionadas de manera centralizada, lo que, aunque aporta flexibilidad y escalabilidad, también introduce nuevos vectores de ataque potenciales.

Las amenazas específicas para las SDN pueden variar desde ataques de denegación de servicio (DoS), la manipulación de la capa de control, hasta la introducción de flujos de datos maliciosos. En consecuencia, la seguridad en las SDN es crítica y requiere estrategias de defensa robustas y adaptativas. Algunas amenazas presentes en las SDN según [2] pueden categorizarse de acuerdo al objetivo del ataque, como por ejemplo, la interceptación en una interfaz de control puede etiquetarse como un ataque que tiene como objetivo principal alterar información provada y sensibles intercambiados entre los dispositivos de red, esto representa una divulgación ilícita de información en un sentido más amplio. algunos problemas de seguridad en las SDN se muestran en el siguiente mapa en la figura 1.5

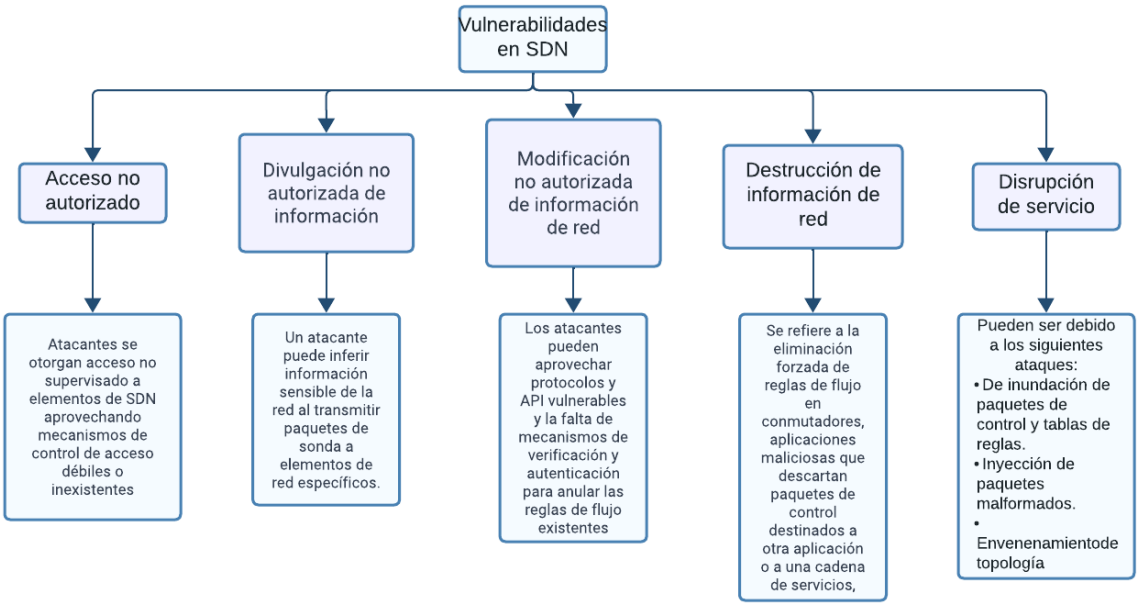


Figura 1.5: Mapa de los principales problemas de seguridad de la arquitectura SDN

Todas las capas e interfaces de la arquitectura de una SDN son sensibles a ataques, podemos identificar los ataques según la capa en donde este se realiza, por ejemplo, en la capa de aplicación se pueden tener las siguientes amenazas:

- Terminación de la aplicación mediante el abuso de privilegios y autoridad fijos,
- neutralización de servicios en donde las aplicaciones maliciosas exitosamente

instaladas en el controlador pueden ser utilizadas para manipular los controladores de paquetes de control

- Ataques a las APIs Northbound vulnerables en donde las malas configuraciones y vulnerabilidades pueden ser aprovechadas para terminar una aplicación víctima emitiendo un comando de sistema o para exponer información intercambiada entre el controlador y una aplicación objetivo

Para detectar y mitigar las posibles amenazas se han propuesto muchos metodos y técnicas, entre las mas eficientes que se han presentado en los últimos años han sido aquellas que utilizan inteligencia artificial para reconocer patrones que no son fácilmente detectables por los operarios de la red. Se han desarrollado modelos utilizando diferentes técnicas y algoritmos de inteligencia artificial para obtener el modelo mas eficiente. [15] utiliza aprendizaje supervisado para generar un modelo que prediga cuando la red está sufriendo un ataque, dicho modelo mide la cantidad de mensajes **PACKET_IN** que ingresan al controlador, en intervalos de tiempo, para posteriormente utilizar los valores de los 10 intervalos como datos de entrada en la predicción. Tang, Wang y Yan en [16] realizan la detección de ataque en dos fases. Durante la primera etapa, la identificación de anomalías basada en el tráfico de los puertos se encarga de determinar si ha sucedido algún evento atípico en un switch, analizando el equilibrio entre el tráfico que entra y sale del switch que ha sido víctima, y que está conectado al punto de cuello de botella. En la segunda etapa, la detección basada en las estadísticas de tráfico de la tabla de flujos necesita identificar con precisión los ataques LDoS (Los ataques LDoS envían ráfagas intermitentes de alta velocidad para deteriorar el rendimiento del servicio de la víctima con un consumo mínimo de recursos) a través de la clasificación. Específicamente, primero.

Tok y Demirci en [17] determina las vulnerabilidades que tiene el protocolo DHCP en algunos controladores para SDN, concluyen que son vulnerables a ataques de agotamiento, y que las inundaciones de mensajes de descubrimiento de DHCP también pueden usarse para lanzar ataques de denegación de servicio, por ejemplo inundando al servidor con mensajes de descubrimiento ilegítimos, lo que puede resultar en un agotamiento de direcciones y en el colapso del servidor inhabilitándolo para recibir solicitudes legitimas y ralentizando las redes y sobrecargando los

controladores. Este análisis se realiza en los controladores POX, ONOS y FloodLight, su método a grandes rasgos identifica a los usuarios legítimos para que puedan acceder al servicio DHCP.

Los ataques de denegación de servicios (Denial of service o simplemente DOS) han evolucionado al igual que las técnicas para mitigarlas, hoy en día estos ataque se lanzan desde diferentes puntos en la red, a estos ataque se los conoce específicamente como ataques de denegación de servicios distribuidos (DDOS) y son el tipo de ataque mas común. Los registros muestran que solo en el primer trimestre de 2013 el ancho de banda de ataque promedio superó los 48.25 Gbps, lo que es aproximadamente un 700 % más que el ancho de banda consumido en el último trimestre de 2012[18]. En 2022, el número de ataques DDoS aumentó un 150 % a nivel global comparados con el año anterior. El número de ataques en América creció aún más rápido, aumentando un 212 % en comparación con 2021. Más de la mitad de los ataques estuvieron dirigidos a organizaciones en EMEA (Eurpoa, Medio Oriente y África). América representó el 35 % de los ataques, mientras que el 7 % de los ataques se dirigieron a organizaciones de APAC (Asia y pacifico). Los reportes de amenazas y ataques a las redes en Ecuador, sobretodo a nivel corporativo se ha mantenido a lo largo de los ultimos meses, pero a finales de agosto hubo una gran cantidad de reportes de ataques, como se puede distinguir en la figura 1.6

Figura 1.6: Número de notificaciones de ataques detectados.

1.6. Aprendizaje de máquina

El machine learning, aprendizaje automático o aprendizaje maquina es un enfoque de la inteligencia artificial que implica la capacitación de sistemas

informáticos para mejorar automáticamente su eficiencia en una tarea específica a medida que se exponen a más datos. En lugar de programar explícitamente reglas y algoritmos, el aprendizaje automático permite a las máquinas aprender patrones y tomar decisiones basadas en la experiencia previa, lo que les permite tomar acciones de manera más precisa y eficiente a medida que adquieren información. el aprendizaje a partir de ejemplos, ha sido un área de investigación activa durante varias décadas [19].

Existen diversas investigaciones en donde se utilizan diferentes tipos de inteligencia artificial para detección de ataques, como por ejemplo Mukkamala, Janoski y Sung en [20] describen enfoques para la detección de intrusiones utilizando redes neuronales y máquinas de vectores de soporte. Otros autores como Comaneci y Dobre [21] y Nisharani , Narayan y Baligar [22] realizan la clasificación de flujos de tráfico mediante aprendizaje supervisado y no supervisado, clasificando las conexiones como legitimas y no legitimas.

1.6.1. Características

Las características del aprendizaje automático incluyen utilizar datos para capacitar modelos que puedan hacer predicciones o tomar decisiones en función de esos datos. Esto implica la capacidad de generalizar a partir de ejemplos de entrenamiento para manejar nuevas situaciones, y la posibilidad de ajustar los modelos según la retroalimentación proporcionada por su rendimiento en datos no vistos. Además, el aprendizaje automático puede abarcar diversos enfoques, como el aprendizaje supervisado, donde se suministran respuestas etiquetadas, y el no supervisado, que busca encontrar patrones en datos sin etiquetas. La elección del algoritmo correcto y la evaluación adecuada de los modelos son aspectos clave en el proceso de aprendizaje automático.

El uso de estas características permite anticiparse a situaciones adversas que puede sufrir la red, como las citadas en la sección anterior. sin embargo aun quedan muchas situaciones por analizar y para tratar de encontrar una solución óptima. Muchas soluciones propuestas generan un modelo con los datos obtenidos del comportamiento real de la red, pero si bien puede cubrir un amplio rango

de posibilidades de estado de la red no serian tan flexibles si el cambio en el comportamiento o en la topología de la red es drástico. [23] propone en su trabajo realizar una revisión continua del algoritmo para modificar los pesos necesarios.

1.6.2. Tipos de aprendizaje supervisado

Existen varios tipos de aprendizaje supervisado que se adaptan a diferentes tipos de problemas y datos. Algunos de los más comunes son:

- Clasificación: En este tipo de aprendizaje supervisado, el objetivo es designar a una categoría o etiquetar a una instancia. Por ejemplo, clasificar un tipos de rosas, o identificar el tipo de objeto en una imagen (por ejemplo, gato, perro, coche).

- Regresión: En lugar de predecir una categoría, en la regresión se trata de predecir un valor numérico. Por ejemplo, predecir el precio de alguna acción de acuerdo al desempeño de la empresa o predecir la temperatura basada en factores climáticos.

- Detección de Anomalías: En este tipo, el objetivo es identificar instancias que son inusuales o atípicas en relación con el resto de los datos. Esto es útil para detectar transacciones fraudulentas en tarjetas de crédito o fallos en sistemas.

- Segmentación: La segmentación implica dividir un conjunto de datos en grupos o segmentos, donde los elementos de una misma categoria son similares entre sí. Esto se usa en análisis de clientes, agrupando usuarios con características similares.

- Clasificación Multietiqueta: En problemas de clasificación multietiqueta, una instancia puede pertenecer a más de una categoría. Por ejemplo, clasificar noticias en categorías múltiples como deportes, política y entretenimiento.

- Clasificación Multiclase: Similar a la clasificación, pero en este caso una instancia se clasifica en una de varias categorías posibles. Por ejemplo, predecir el tipo de fruta basado en sus características.

Capítulo 2

Diseño e implementación del proyecto

Este capítulo presenta el desarrollo del modelo de detección de ataques. En la primera etapa se presenta el diseño de la topología con la que se trabaja, la instalación y configuración de las herramientas de software requeridas como mininet, Ubuntu, Ryu, Mobaxterm, etc, como segunda etapa se realiza la captura de los datos para posteriormente generar el dataset para el entrenamiento, cabe resaltar que adicionalmente al dataset obtenido mediante la red propia también se ocupan datasets externos y como tercera etapa del desarrollo se realiza el entrenamiento para obtener el modelo. Esta última etapa se la realiza para diferentes datasets que básicamente varían en la cantidad de entradas que tienen, es decir, se realizan modelos para unos cuantos cientos de entradas y luego para unas miles de entradas, para poder realizar comparativas en la eficiencia del modelo, con el propósito de determinar si para este sistema, una mayor cantidad de información genera un modelo mas preciso o si no existe diferencia. En la figura 2.1

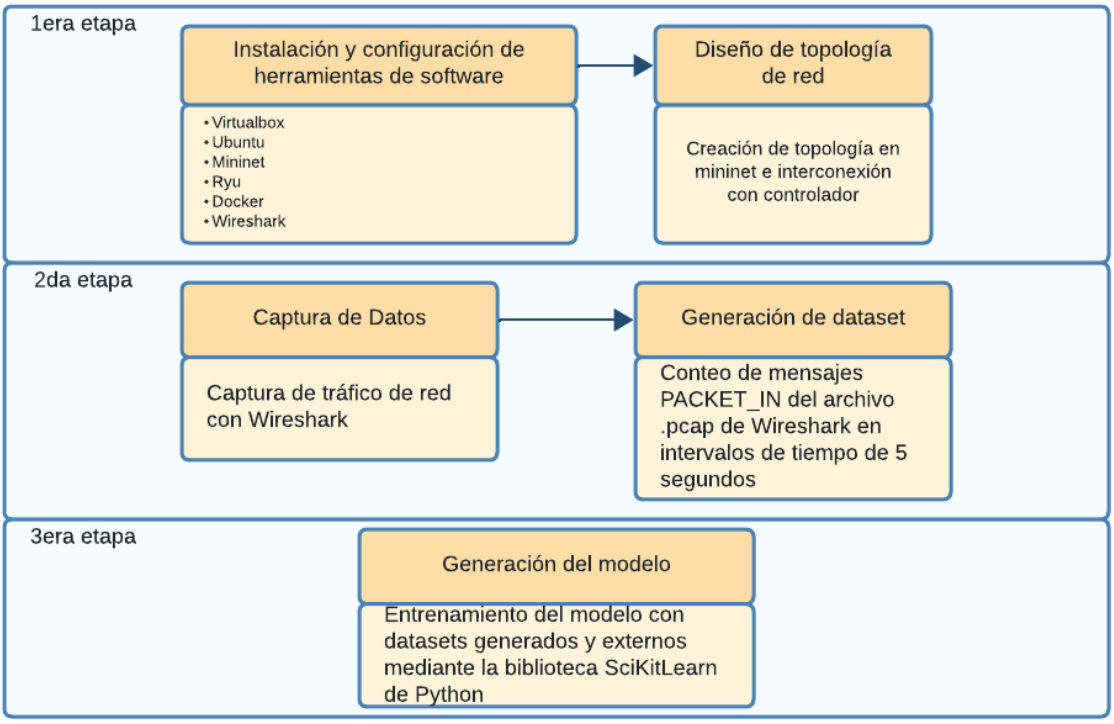


Figura 2.1: Diagrama de las etapas del desarrollo del proyecto.

Los ataques se van a realizar a nivel de la capa de control, buscando saturar al mismo para inhabilitar su respuesta al momento de que un host o un switch necesite establecer una nueva ruta. Se ha mencionado en el marco teórico que en las SDN se centraliza el control de la red, por lo que inicialmente un switch no decide por si solo como reenviar los paquetes, esto lo decide el controlador, pero una vez establecida la ruta, no necesita nada mas del controlador hasta que encuentre alguna nueva solicitud de ruta o cuando tiene alguna inconsistencia o problema con su tabla de flujo, estos paquetes se llaman **PACKET_IN**. Por lo tanto la idea es saturar al controlador con tantos mensajes **PACKET_IN** como sea posible, de modo que no pueda responder a otros nodos legítimos de la red

2.1. Instalación y configuración de software

Para la implementación de la simulación se necesitan varias herramientas con el fin de implementar un escenario lo mas realista posible. Las principales aplicaciones

de software utilizadas se muestran en la tabla 2.1 describiendo la función que tuvo cada una de ellas en el proyecto.

Tabla 2.1: Herramientas de software utilizadas y su función en el desarrollo del proyecto.

Software	Función
Virtualbox	Crear y ejecutar el servicio virtualizado mininet y la máquina virtual en donde se ejecuta el controlador
Mininet	Software en donde se crea y ejecuta la topología de red
Python	Lenguaje de programación para la implementación de la topología con caracteristicas especificas como ancho de banda de enlaces y retardos y para la creación del modelo de aprendizaje supervisado para la detección de ataques
Mobaxterm	Escritorio remoto que proporciona el entorno gráfico para la ejecución de los terminales de los nodos de la red de mininet
Ubuntu	Sistema operativo en donde se ejecuta el controlador Ryu mediante Docker
Ryu	Software de controlador que gestiona y controla el tráfico de la red simulada
Docker	Plataforma para instalar el controlador RYU en un contenedor. Su trabajo es dimensionar las capacidades del controlador de acuerdo a las exigencias de la red propuesta

La instalación de virtual box, que fue el software de virtualización utilizado sobre el que corre ubuntu y mininet, fue bastante sencilla, se descargó la aplicación desde la pagina oficial para windows 10 (software anfitrión) *https://www.virtualbox.org/wiki/Downloads* y se instaló en la forma estándar. La versión utilizada fue la 6.1

2.1.1. Instalación y configuración de mininet

En mininet es en donde se crea y corre la topología de prueba. Existen diferentes formas de instalar este simulador de redes, para este proyecto se utilizó un servicio virtualizado, por la ventaja de que viene preconfigurado.

La topología propuesta se muestra en la figura 2.2. La topología se describe con mayor detalle en la siguiente sección, en la implementación de la red.

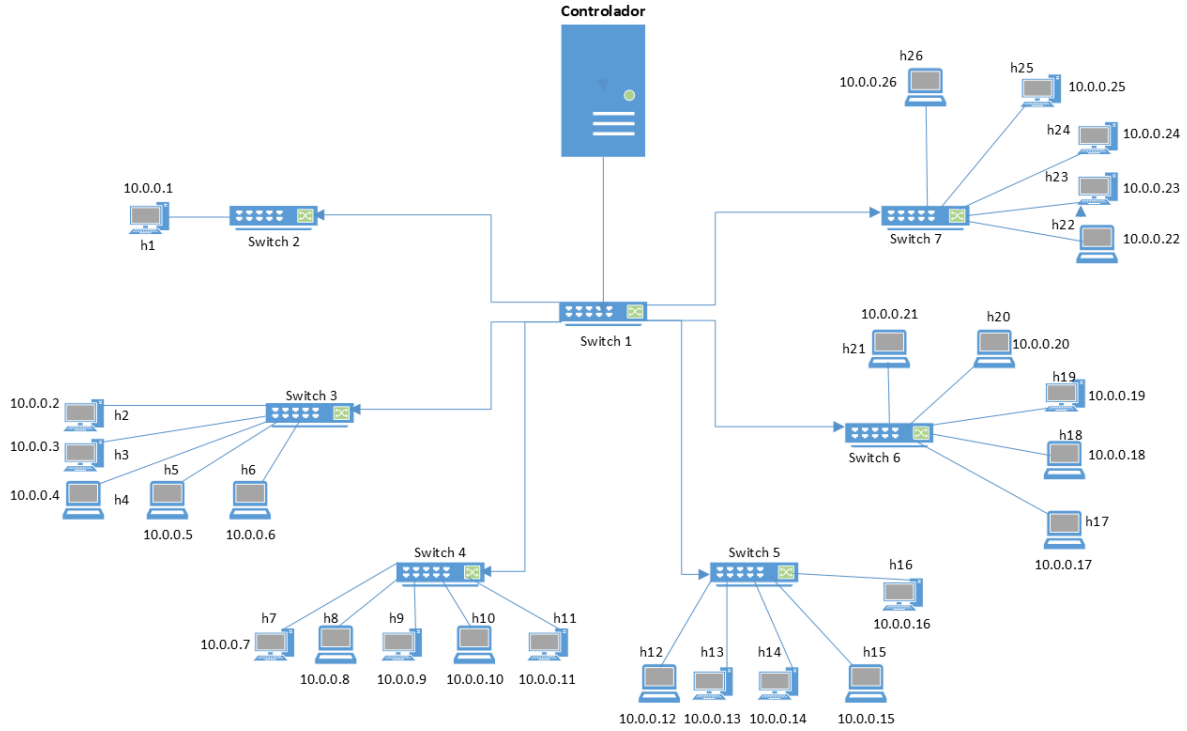


Figura 2.2: Topología de prueba para ataques de denegación de servicios

Para instalar mininet en virtual box, desde la pestaña *archivos* se elige la opción *importar servicio virtualizado* o presionando conjuntamente *ctrl+I*. Allí se abrirá una ventana para la selección del archivo con extensión **ovf** de mininet. La ventaja de instalar un servicio virtualizado es que ya viene preconfigurada la cantidad de memoria base, procesador, almacenamiento, etc. Luego basta con ejecutar la maquina para iniciar. Un cambio necesario en esta máquina virtual fue configurar el adaptador de red principal en una *red NAT* y habilitar una segunda interfaz configurada en *bridge* o *modo puente* para poder acceder a mininet desde el sistema operativo anfitrión y para que se pueda realizar la comunicación remota entre el controlador que se ejecuta en Ubuntu y la red simulada.

Mininet se maneja en linea de comandos y para el ingreso solicita un usuario y contraseña que por defecto son *mininet* para los dos casos. la figura 2.3 muestra la interfaz que maneja el software, que consiste en una ventana de comandos.

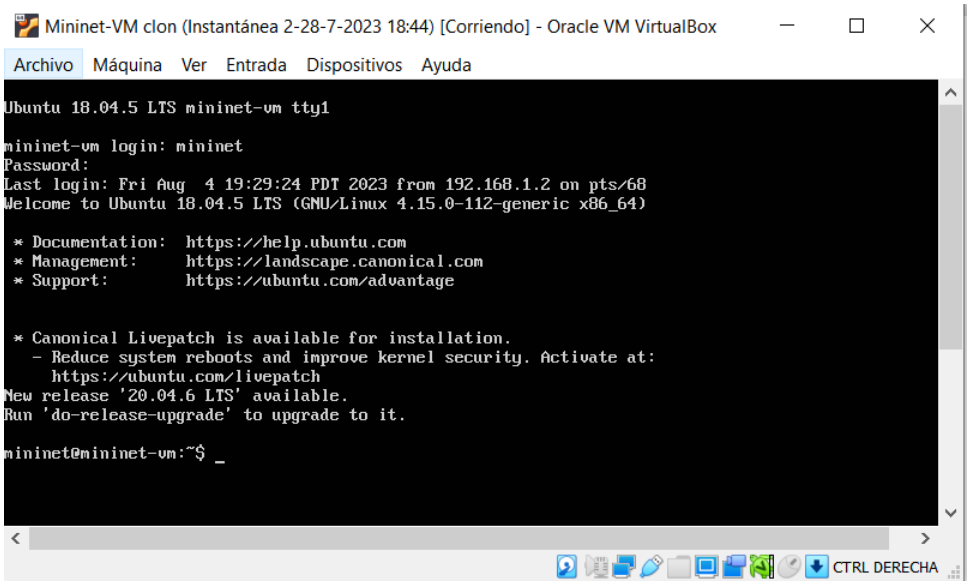


Figura 2.3: Ventana de comandos de mininet.

Mininet no provee directamente de una interfaz gráfica, por lo que al ejecutar la herramienta *miniedit* para crear las topologías el sistema muestra un error. Para solucionarlo se puede utilizar una terminal que provea de interfaz gráfica como *Mobaxterm*. Sin embargo, se pueden crear topologías desde la linea de comandos de mininet e incluso generar topologías personalizadas mediante la biblioteca de python también llamada *mininet*, este último fue el método utilizado para crear la topología de prueba.

2.1.2. Instalación y configuración de Mobaxterm

Este software se descargó desde la página oficial en el siguiente enlace: *https://mobaxterm.mobatek.net/download.html*

La instalación se realizó de forma estándar, sin ningún parámetro adicional. Con esta herramienta se accede a mininet, con la ventaja que desde este software si se pueden ejecutar aplicaciones que requieren de interfaz gráfica como por ejemplo *miniedit*. La conexión desde *mobaxterm* a *mininet* se realizó mediante el protocolo SSH. por lo que se necesitaba conocer de antemano la dirección IP de la máquina virtual de *mininet*. La interfaz gráfica de *mobaxterm* es como se observa en la figura 2.4.

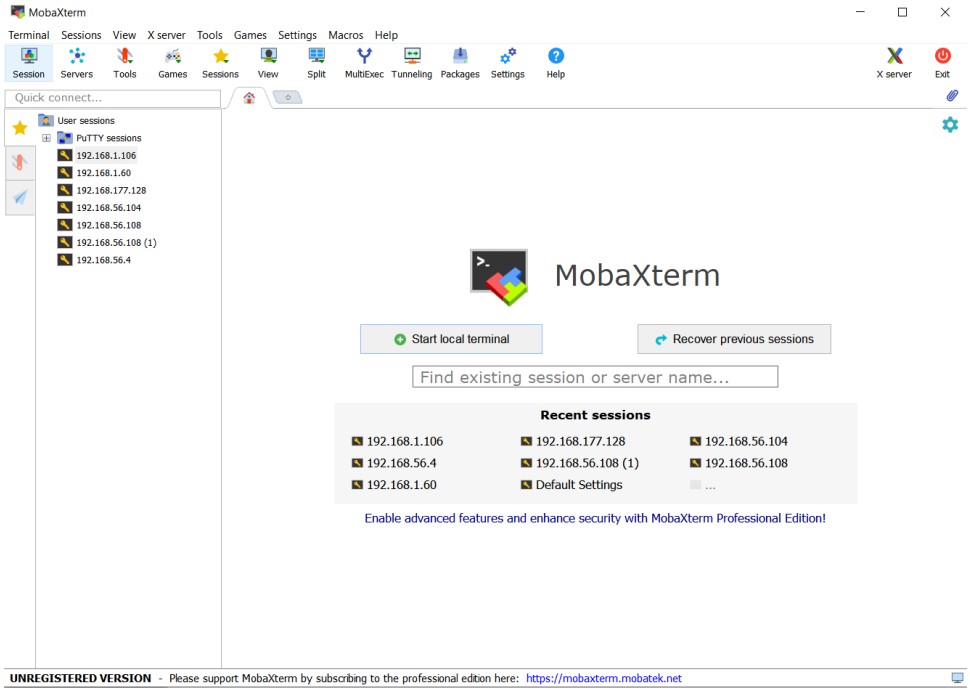


Figura 2.4: Interfaz gráfica de Mobaxterm

Para conectar *mobaxterm* a *miniedit*, este último debe encontrarse en ejecución y se necesita conocer la IP del adaptador que está en modo puente (*eth1*) con el comando **ifconfig**; inicialmente este adaptador no tenia asignada una dirección IP, por lo que fué necesario ejecutar el comando **sudo dhclient eth1** para que el servidor DHCP asigne una dirección, se puede verificar que se ha asignado correctamente una dirección ejecutando nuevamente el comando *ifconfig*. La conexión se realizó accediendo a la pestaña *Session* en la esquina superior izquierda de *mobaxterm*, luego, seleccionando la opción *SSH* aparece la ventana con los parámetros necesarios para iniciar sesión como se muestra en la figura 2.5, aquí se ingresa la dirección IP de *mininet* y el puerto que por lo general viene por defecto el 22.

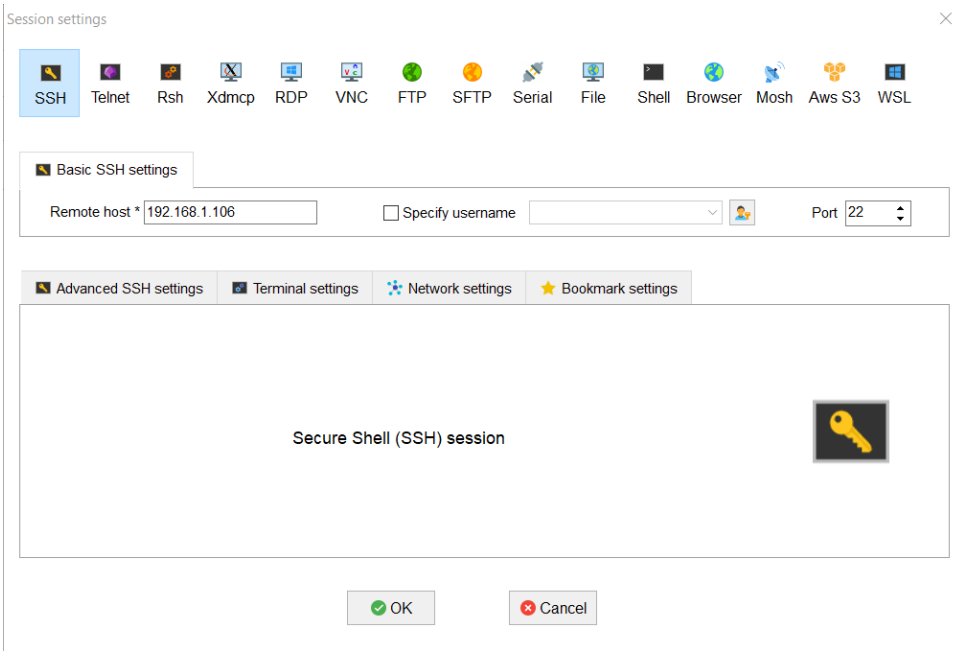


Figura 2.5: Inicio de sesión SSH entre mobaxterm y mininet.

Para ingresar, pide la contraseña de mininet que como se mencionó anteriormente, es **mininet** por defecto. Una vez dentro, podemos trabajar como normalmente se hace en la ventana de comandos de *mininet*.

La topología se la realizó mediante un código en Python debido a la facilidad para personalizar algunos parámetros como ancho de banda y retardo en los enlaces, direcciones IP de los host, ejecución de scripts de los host, entre otras cosas, pero para fines prácticos, la topología fue implementada en *miniedit* para obtener un contexto gráfico de la red que se utiliza para las pruebas.

Miniedit se encuentra en el directorio *mininet/examples*. Para ejecutarlo se lo puede hacer directamente desde la carpeta actual colocando toda la ruta o primero ingresando a la carpeta con *cd* y luego ejecutando, como se indica a continuación.

```
# Ejecutando mininet desde el directorio actual:
sudo python mininet/examples/miniedit.py
# Ingresando en la carpeta llamada "examples"  y luego ejecutando miniedit
cd mininet/examples
sudo python miniedit.py
```

2.1.3. Instalación y configuración de Ubuntu, Docker y RYU

La imagen ISO de ubuntu para la maquina virtual se obtuvo de la pagina oficial en *https://ubuntu.com/download/desktop*. La maquina virtual fue creada con las siguientes caracteristicas:

- Nombre del SO: Ubuntu
- Tipo: Linux
- Versión: Ubuntu (64-bit)
- Capacidad de memoria: 2368 Mb
- Disco virtual: 54 Gb

Al igual que con *mininet*, el primer adaptador de red se creo conectado a *NAT* y el segundo adaptador habilitado en *modo puente*, con el proposito de permitir que la maquina virtual de *mininet* y *Ubuntu* se puedan comunicar, así como *Ubuntu* y el sistema operativo anfitrión.

Una vez creada la maquina virtual se realizó la instalación seleccionando la imagen de Ubuntu previamenten descargada. La instalación se realizó con los parámetros por defecto de *Ubuntu*

En ubuntu se realizó la instalación de *Docker* para poder correr un contenedor que contenga *RYU*, el propósito de *Docker* es limitar la capacidad computacional del controlador debido a que se está trabajando con una red relativamente pequeña, por lo que es necesario dimensionar las capacidades del controlador al entorno de red para observar como afectan los ataques a estos recursos limitados, además al trabajar con *Docker*, es posible acceder a varias instancias de controlador, lo que permite utilizar varias ventanas para las diferentes tareas al momento de capturar el trafico y ejecutar los códigos de Python.

La instalación de *Docker* se realizó desde la terminal de *Ubuntu* siguiendo estos pasos:

```
# Actualización de paquetes existentes
sudo apt-get update
# Instalación de paquetes para HTTPS
sudo apt-get install apt-transport-https ca-certificates curl
software-properties-common
# Clave GPG para el repositorio oficial de Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
# Añadir el repositorio de Docker a las fuentes de APT
sudo add-apt-repository
"deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
# Actualización
sudo apt-get update
# Verificación del repositorio de Docker
sudo apt-cache policy docker-ce
# Instalación de Docker
sudo apt-get install docker-ce
# Verificación de instalación
sudo docker --version
```

La versión utilizada de *docker* es la 24.0.4. La instalación del controlador *Ryu* en *Docker*, se hizo con la siguiente linea de código.

```
docker pull osrg/ryu
```

2.2. Ejecución del controlador RYU

Primero se ejecuta el controlador para que este se encuentre a la escucha de las solicitudes de los nodos de la red.

```
sudo docker run --net=host --it --rm --name ryu --cpus=".2"
--memory="10000m" osrg/ryu
```

En el código anterior el parámetro **–net=host** le indica a *Docker* que el contenedor tome una dirección en el mismo rango de red que la máquina anfitrión. **–name ryu** asigna el nombre *ryu* al contenedor, **–cpus=".2"** le indica al contenedor que solo puede usar el 20 % de la CPU y **–memory="1000m"** le indica la cantidad de RAM que puede usar.

Dentro del contenedor en ejecución se implementa un switch para para posteriormente ejecutar la topología en *mininet*. Se ejecuta la aplicación *simple_switch_13* que es un switch que utiliza la versión 1.3 de *OpenFlow*. Esta aplicación permite manejar paquetes de red y crear flujos de conmutación en los switches de red; para ejecutar el comando se ingresa a la carpeta app y se ejecuta la aplicación.

```
cd ryu/ryu/app
ryu-manager simple_switch_13.py
```

Esta aplicación permite observar los paquetes que pasan por los switches en el controlador.

2.3. Implementación y ejecución del diseño del diseño de red

La propuesta de la RED de prueba se realizó en la API de python de *mininet*, pero para facilitar el reconocimiento de los nodos, se realizó un esquema que se mostró en la figura 2.2. la figura 2.6 muestra un diagrama de flujo para con los pasos que se realizan para ejecutar toda la simulación

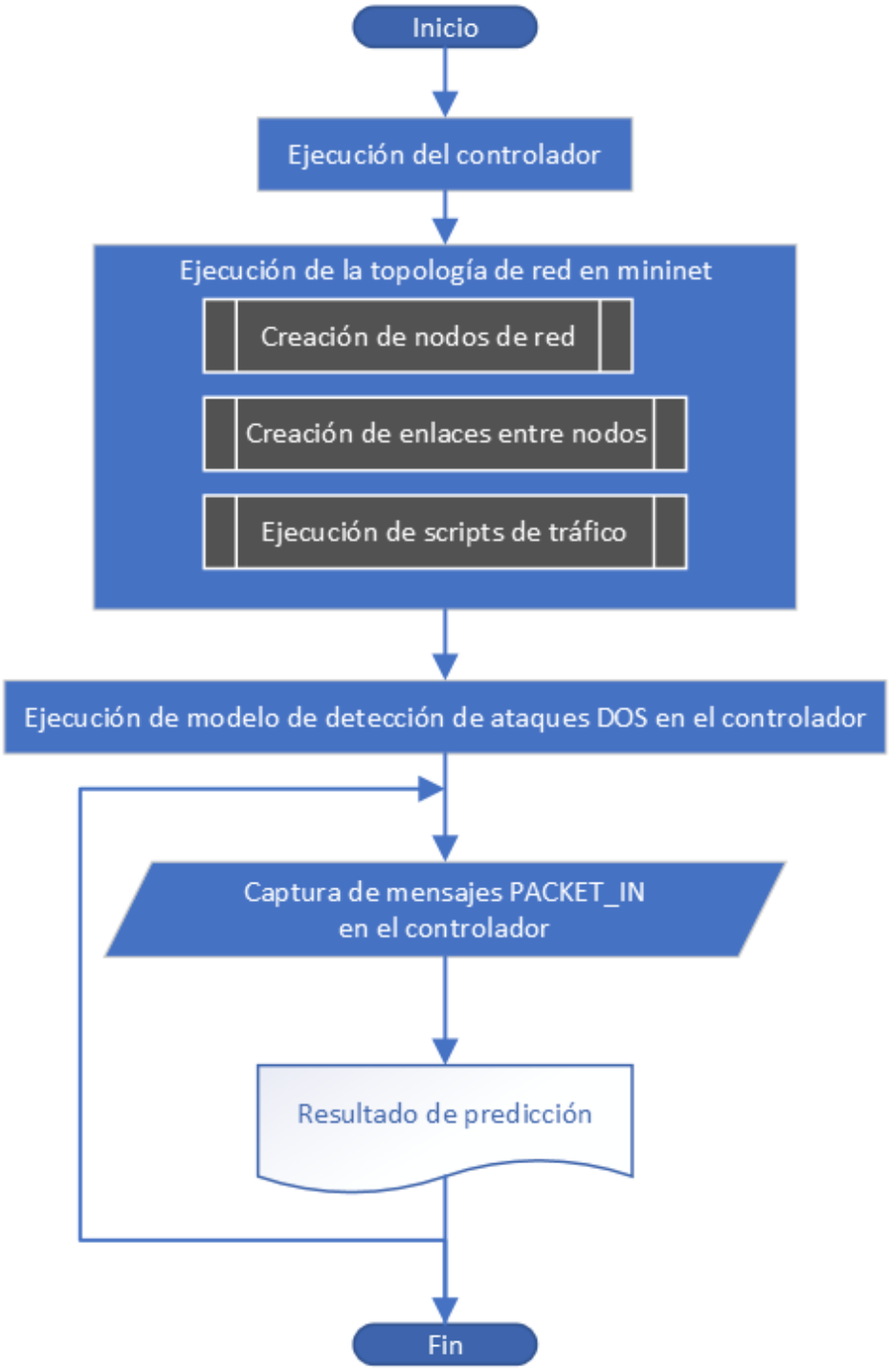


Figura 2.6: Diagrama de flujo para le ejecución de la simulación.

La figura 2.7 muestra la topología que se implementó con la herramienta *miniedit* de *mininet*; si bien la topología es implementada mediante codgio en Python, este paso de crear la topología en *miniedit* ayuda a verificar inicialmente que la red simulada tiene conectividad con el controlador remoto. Aquí se se tiene un controlador

remoto, 7 switches y 26 hosts. El switch **s1** se conecta físicamente al controlador **c0** y a los switches **s2** al **s7**. El host 1 **h1** se conecta a **s1** y de ahí se conectan 5 hosts a cada uno de los otros switches.

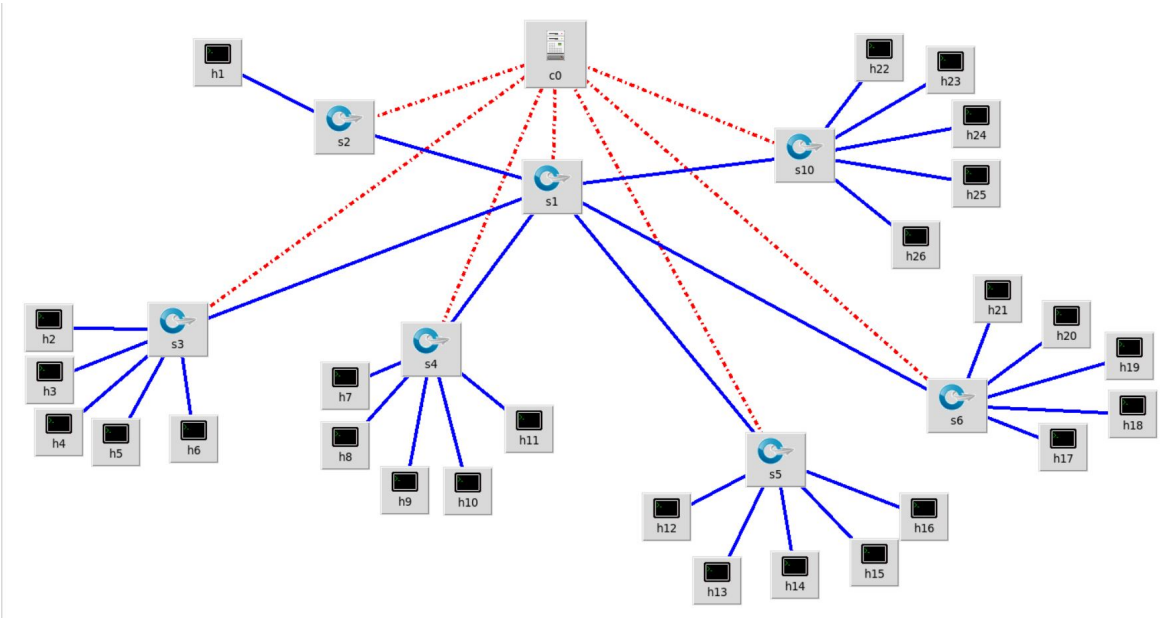


Figura 2.7: Topología de prueba para ataques de denegación de servicios

La asignación de las direcciones IP se lo realiza siguiendo el orden de numeración que tiene cada host, por ejemplo el host 1 tiene la IP 10.0.0.1 y el host 26 la dirección 10.0.0.26, esto se hace para recordar con facilidad las direcciones al momento de ejecutar las pruebas.

La topología implementada en Python se ha dividido y se muestra en los siguientes bloques código. Se explica a grandes rasgos cada sección del código.

Se importan todas las librerías necesarias para los switches, que para este trabajo es el OVSK, para la consola, enlaces, de tiempo para generar retardos y las demás bilbiotecas de mininet.

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

Se define una función, llamada 'UPSTopoTesis', aquí se definen el tipo de controlador que es remoto en este trabajo, se crean los hosts, enlaces, retardos, etc. La libreria TCLink sirve para definir el ancho de banda y los retardos en los enlaces entre los nodos. La dirección del controlador remoto **192.168.56.108** es la IP de la máquina virtual Ubuntu, que es en donde se está ejecutando.

```
def UPSTopoTesis():
        "Topologia propuesta"
        net = Mininet(controller=RemoteController,link=TCLink)

        info('*** Anadiendo controlador\n')
        c0 = RemoteController('c0',ip='192.168.56.108',port=6633)
        net.addController(c0)
```

El comando *info* se utilizó en cada segmento del código para conocer en que parte de la ejecución se encuentra, en caso de que llegue a darse algún error. Se muestra únicamente el código para la creación de 4 hosts con uss respectivas direcciones IP, ya que este proceso se repite para los otros 22 hosts.

```
info('*** Anadiendo hosts\n')
        h1 = net.addHost('h1',ip='10.0.0.1')
        h2 = net.addHost('h2',ip='10.0.0.2')
        h3 = net.addHost('h3',ip='10.0.0.3')
        h4 = net.addHost('h4',ip='10.0.0.4')
```

Una vez creados los hosts, se crean los switches, en donde se indica las clase de switch (*OVSK*) y el protocolo utilizado (*OpenFlow*). Este código se repite par los otros 4 switches.

```
info('*** Anadiendo switchs\n')
        s1 = net.addSwitch('s1',cls=OVSKernelSwitch,protocols='OpenFlow13')
        s2 = net.addSwitch('s2',cls=OVSKernelSwitch,protocols='OpenFlow13')
        s3 = net.addSwitch('s3',cls=OVSKernelSwitch,protocols='OpenFlow13')
```

Con los hosts y switches creados, queda enlazarlos; aquí se debe definir el ancho de banda de cada enlace y el retardo o delay en la transmisión. La topología creada en *miniedit* resulta de utilidad aquí para conocer los nodos que tienen enlaces. Los enlaces entre switches se crearon con un ancho de banda de 1 Gbps y entre hosts y switches entre 100 Mbps y 1 Gbps, con esto lo que se busca conseguir es un escenario mas apegado a una red real. En el código solo se muestra 4 enlaces. El código completo se adjunta al final de este documento.

```
info('*** Creando enlaces')
        net.addLink(s1,s2,bw=1000,delay='1ms')
        net.addLink(s1,s3,bw=1000,delay='1ms')
        net.addLink(h2,s3,bw=500,delay='5ms')
        net.addLink(h3,s3,bw=100,delay='5ms')
```

Se ejecuta la red.

```
info( '*** Iniciando red\n')
        net.start()
```

2.3.1. Simulación de tráfico

La red como tal no tiene nodos transmitiendo tráfico. Para simular trafico se crearon scripts que usan la herramienta **Iperf**. El trafico generado es del tipo TCP y UDP periódico y aleatorio, en cuatro scripts que van a ser ejecutados por la mayoría de hosts de la red:

- Trafico TCP periódico que utiliza el máximo ancho de banda del enlace, este código envía tráfico cada 40 segundos.
- Tráfico TCP en tiempo aleatorio, entre 10 y 60 segundos.
- Tráfico UDP periódico, enviando 100 bytes cada 10 segundos.
- Trafico UDP en tiempo aleatorio entre 40 y 100 segundos

```
info('*** Ejecutando scripts iniciales de trafico')
        h1.popen('/home/mininet/mininet/examples/tesis/ServIperf.sh')
        h2.popen('/home/mininet/mininet/examples/tesis/TCPTrafPeriodico.sh')
        h4.popen('/home/mininet/mininet/examples/tesis/UDPTrafPeriodico.sh')
        h6.popen('/home/mininet/mininet/examples/tesis/TCPTrafAleatorio.sh')
        h8.popen('/home/mininet/mininet/examples/tesis/UDPTrafAleatorio.sh')
```

Finalmente se ejecuta la ventana de comandos *CLI* y se implementa el método principal.

```
info( '*** Running CLI\n' )
        CLI( net )
        info( '*** Stopping network' )
        net.stop()
if __name__ == '__main__':
        setLogLevel( 'info' )
        UPSTopoTesis()
```

La topología se ejecuta de la siguiente manera en la consola de *mininet*:

```
sudo python UPSTopoTesis3.py
```

En la interfaz de linea de comandos se ejecuta el comando **nodes** para corroborar que se han creado todos los nodos y el comando **net** para verificar que

las conexiones son correctas. Inicialmente los nodos no conocen los caminos a los otros nodos, para ello se ejecuta el comando **pingall** para que cada nodo encuentre su trayectoria a los otros nodos. La salida del comando *info* del código UPSTopoTesis3.py se puede observar en la siguiente imagen 2.8.

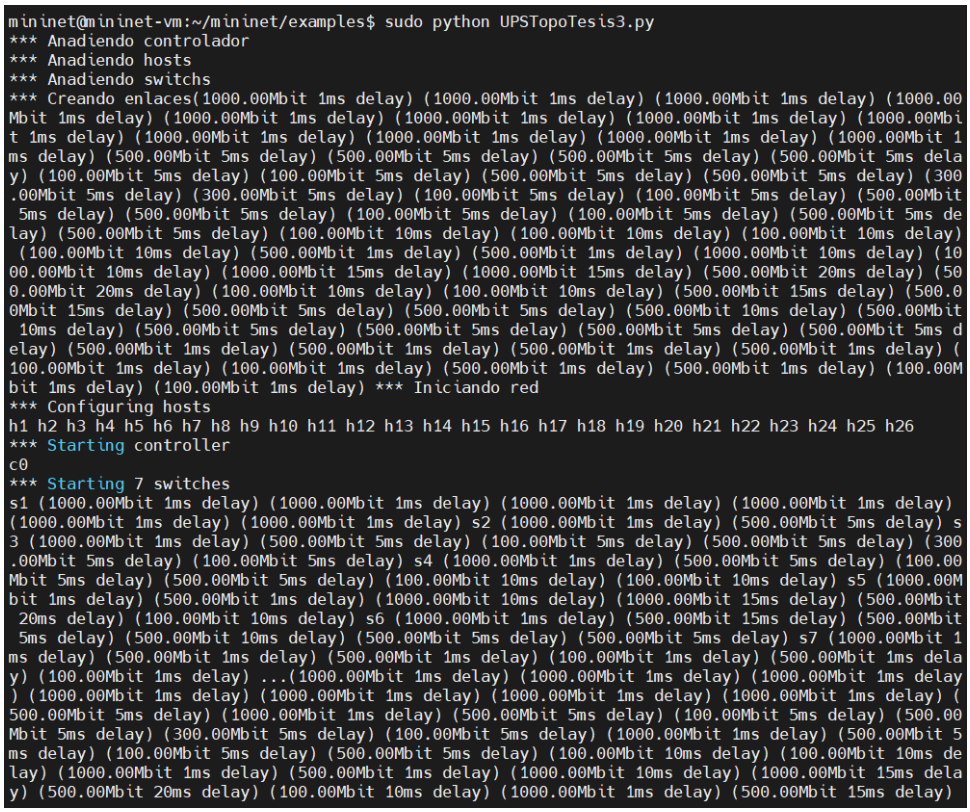


Figura 2.8: Ejecución de la topología de red.

Gracias al comando *info* se puede observar el momento en que se va creando cada nodo y sus enlaces, además de las características de cada enlace, como por ejemplo el ancho de banda entre los enlaces junto con la latencia del enlace, esto se puede observar encerrado entre paréntesis y el orden va de acuerdo a como se agregaron los enlaces en el código de python para la topología.

2.4. Captura de tráfico

2.4.1. Tráfico de red sin ataque

Los nodos de la red se encuentran enviando tráfico TCP y UDP a tiempos regulares y aleatorios, este es el funcionamiento normal de la red. Para capturar el

trafico se utiliza la herramienta *Wireshark* desde el lado del controlador.

Para verificar que la red se encuentra funcionando con normalidad, se realizó una prueba de conectividad *ping* entre todos los diferentes pares de hosts (2.9) con el comando **pingall** de mininet, este comando, como se puede observar en la figura 2.9, realiza un ping entre todas las combinaciones de pares de host que hay en la red, por ejemplo, toma al *host1* y realiza un ping hacia todos los demas host, luego realiza el mismo proceso con el *host2* y así susecivamente hasta completar la prueba con todos los hosts.

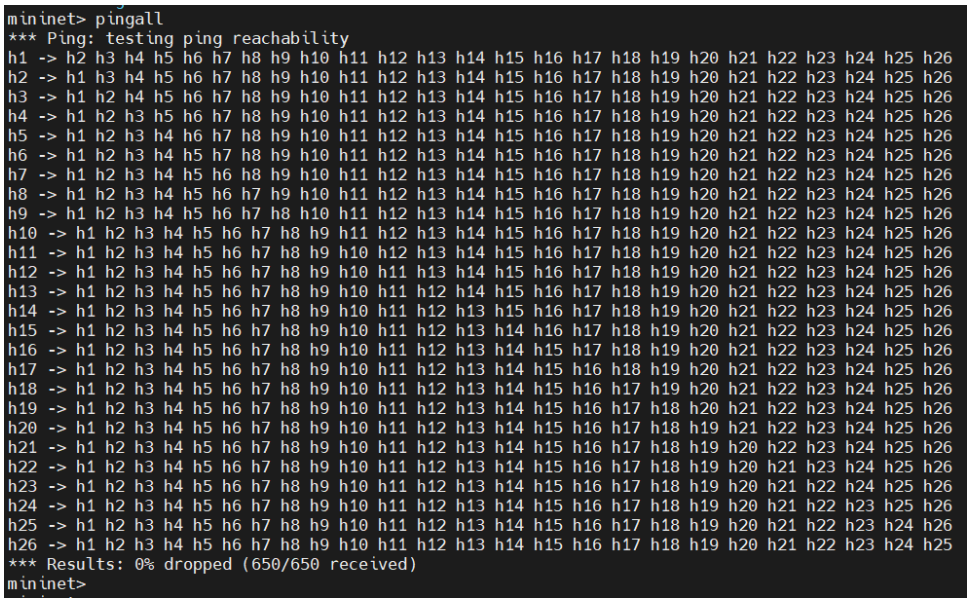


Figura 2.9: Verificación de conectividad entre todos los host

Esto también nos da la certeza que el controlador está ejecutándose correctamente, por que de no ser así, los nodos no podrían alcanzarse.

Una vez que todos los nodos de la red tienen establecida su ruta, no necesitan comunicarse nuevamente con el controlador por este motivo, exceptuando los mensajes de conectividad que mantienen constantemente y otro tipo de mensajes que comparten continuamente los dos. Es decir que los switches no enviarán mas mensajes **PACKET_IN** al controlador, sin embargo, en una red real, muchas rutas se van creando constantemente ya que los nodos se agregan y desagregan en tiempo real. Para simular este comportamiento, los nodos envían trafico a los nodos existentes y a otras direcciones que no son existentes en la red de forma aleatoria, esto genera paquetes **PACKET_IN** en tiempos aleatorios.

En el punto 2.2 se indicó como ejecutar el controlador e implementar un switch *OVSK*, previo a ejecutar la topología en mininet. Inicialmente se puede observar los paquetes que llegan al controlador cuando los nodos están conociéndose y debido el tráfico que envían los host a direcciones fuera de la red, por lo que se espera ver mensajes llegando constantemente como se puede apreciar en la figura 2.10.

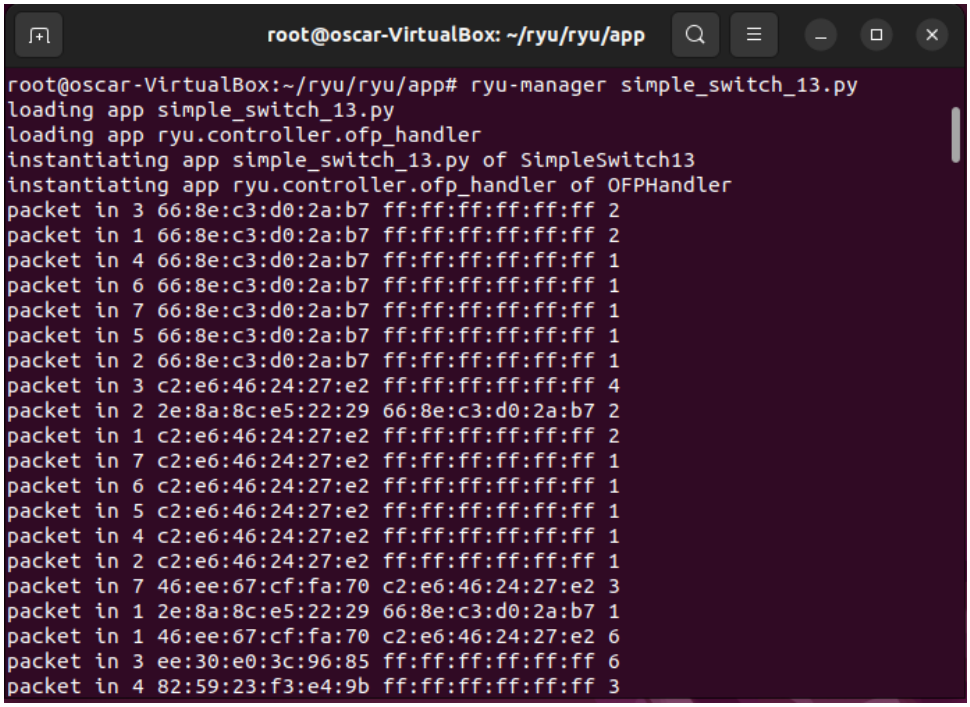


Figura 2.10: Mensajes del tipo *PACKET_IN* llegando al controlador

En una nueva terminal se ejecuta *Wireshark* y en el filtro se especifica que sean únicamente mensajes del tipo **PACKET_IN**. La interfaz seleccionada es la **enp0s8** que corresponde a la que se conecta con la simulación en *mininet*, la dirección IP de esta interfaz es la que se asigna al controlador en la siulación. En la figura 2.11 se pueden observar algunos mensajes que llegan al controlador desde la red.

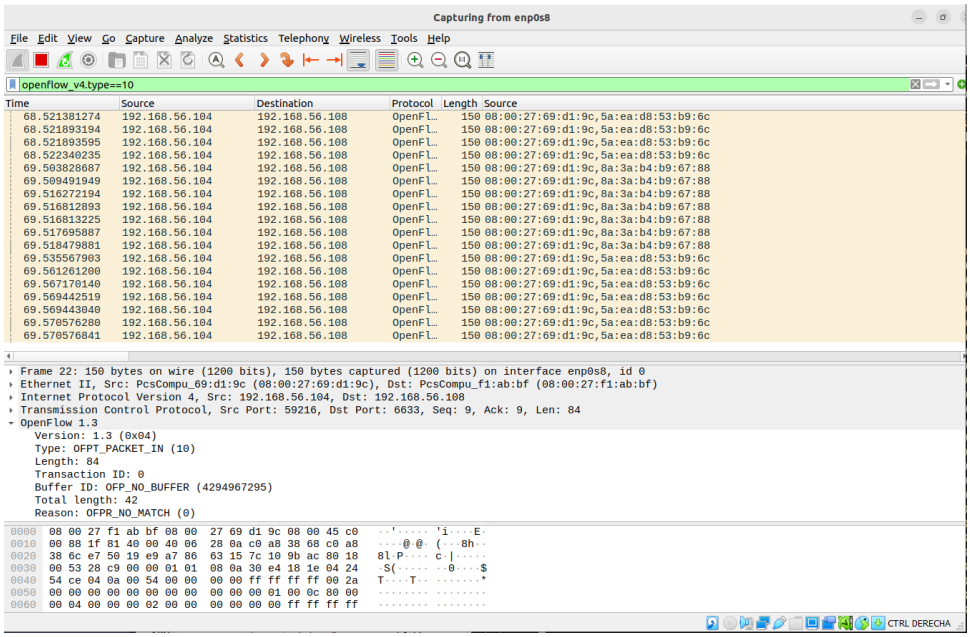


Figura 2.11: Datos capturados con wireshark en la red con tráfico normal

Las sesiones de captura de datos se realizaron a intervalos de tiempo de 2 a 5 horas por día debido a las limitaciones de recursos computacionales de la maquina anfitrión.

2.4.2. Tráfico de red bajo ataque

El tráfico se capturó de la misma manera que para la red en estado normal, con la diferencia de que en este caso varios hosts se encuentran enviando solicitudes de rutas a direcciones inexistentes, lo que provoca que los switches envíen mensajes al controlador para que este determine como proceder y establezca una nueva regla de flujo en los switches. Como se ha mencionado anteriormente, los ataques consisten en enviar la mayor cantidad de mensajes **PACKET_IN** al controlador y esto se consigue enviando paquetes ARP a direcciones aleatorias inexistentes en el menor tiempo posible. En las pruebas, el controlador empezó a colapsar cuando se lanzó el ataque desde cuatro maquinas, como se muestra en la figura. Los host seleccionados para este propósito fueron los impares (h3, h5, h7, h9). Para ejecutar los scripts del ataque se utilizó el emulador de terminal **Xterm**, esta herramienta permite abrir una terminal en un nodo y desde allí se ejecutó el código de Python que contiene el ataque.

```
# Abrir terminal del host atacante en la consola de mininet
> xterm h3
# Dentro de la terminal ejecutar el codigo del ataque
sudo python tesis/Ataque.py
```

La figura 2.12 muestra la salida en consola durante el ataque.

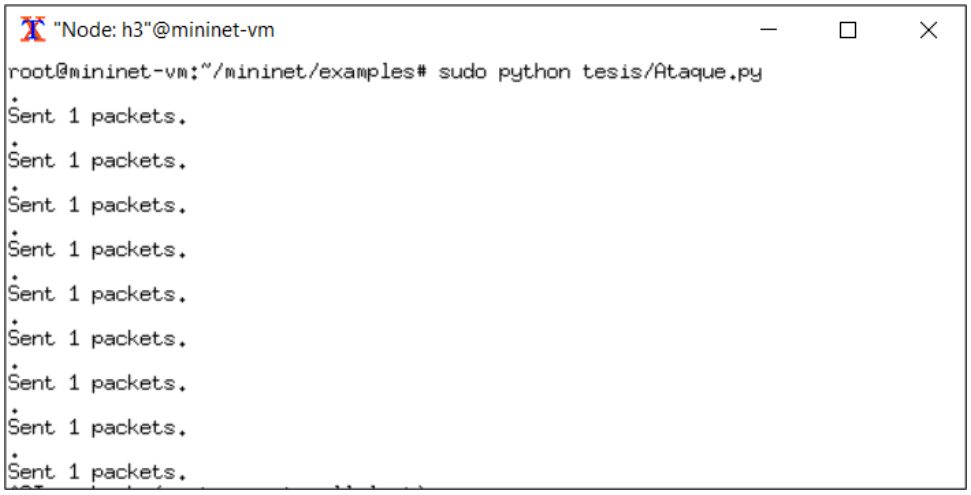


Figura 2.12: Ataque **DOS** lanzado desde el host h3

La captura en wireshark devuelve la misma salida en pantalla pero la cantidad de paquetes recibidos fue mucho mayor que en el caso anterior, con tráfico normal, luego de lanzar el ataque desde los cuatro hosts. La captura se hizo para diferente cantidad de nodos atacantes, desde uno hasta 4 hosts, para que el modelo de detección pueda prevenir el ataque, antes de que este colapse al controlador. Es decir, durante la simulación y captura de tráfico se mantuvo el ataque desde un host durante un lapso considerable de tiempo para luego ir aumentando de uno en uno la cantidad de atacantes, lo que permite obtener atributos con una menor cantidad de tráfico, pero que implica que la red está siendo atacada, el objetivo de este procedimiento es dar al administrador de la red un aviso previo para que pueda mitigar el ataque antes de que sea tarde.

Se puede corroborar que el controlador está saturado enviando un *ping* desde algún host hacia otro nodo del que no conozca la ruta, así como se observa en la siguiente figura 2.13.

```
mininet> h19 ping h26
PING 10.0.0.26 (10.0.0.26) 56(84) bytes of data.
From 10.0.0.19 icmp_seq=1 Destination Host Unreachable
From 10.0.0.19 icmp_seq=2 Destination Host Unreachable
From 10.0.0.19 icmp_seq=3 Destination Host Unreachable
From 10.0.0.19 icmp_seq=4 Destination Host Unreachable
From 10.0.0.19 icmp_seq=5 Destination Host Unreachable
From 10.0.0.19 icmp_seq=6 Destination Host Unreachable
From 10.0.0.19 icmp_seq=7 Destination Host Unreachable
From 10.0.0.19 icmp_seq=8 Destination Host Unreachable
From 10.0.0.19 icmp_seq=9 Destination Host Unreachable
From 10.0.0.19 icmp_seq=10 Destination Host Unreachable
From 10.0.0.19 icmp_seq=11 Destination Host Unreachable
From 10.0.0.19 icmp_seq=12 Destination Host Unreachable
```

Figura 2.13: Controlador colapsado debido a ataque de denegación de servicios

2.5. Estructuración de Dataset

Los archivos generados en Wireshark son de extensión *.csv*. Se han obtenido varias características para cada entrada de la tabla pero la mas importante es la de tiempo, ya que este archivo únicamente contiene mensajes *PACKET_IN* por lo que con este archivo podemos obtener la cantidad de mensajes por intervalos de tiempo.

El dataset se forma tomando la cantidad de mensajes por intervalos de 5 segundos y luego tomando 10 de estos intervalos para formas los atributos o características y finalmente colocando una etiqueta para indicar si se trata de un ataque o no. La etiqueta (0) indica que la entrada no corresponde a un ataque y (1) indica que la entrada corresponde a un ataque.

La reorganización de las entradas se realizó mediante un código en Python que lee el archivo *csv* en un dataframe de la librería *Pandas*, luego lee cada entrada de la columna 'Time', que es la columna de tiempo del archivo de *.pcapng* de Wireshark, para crear intervalos de 5 segundos, posteriormente realiza un conteo de la cantidad de filas o entradas que se encuentran en dicho intervalo y los coloca como columnas o características del dataset resultante y en la columna final coloca la etiqueta o de forma abreviada cuenta cuantos registros del archivo *pcapng* hay cada 5 segundos; este código al final exporta un archivo *.xls*. Debido a que se generaron varios archivos *.xls* porque la captura se realizó en diferentes momentos, fue necesario realizar este proceso para cada uno de ellos para finalmente recopilar todos en un solo archivo

Tabla 2.2: Algunas entradas del dataset.

0	1	2	3	4	5	6	7	8	9	label
42	21	7	0	28	35	14	7	0	0	0
0	21	0	0	0	0	0	0	0	35	0
7	0	35	14	14	0	0	0	0	0	0
35	77	21	35	0	0	0	21	21	49	0
14	14	7	7	14	0	0	7	35	0	0
322	325	326	322	350	322	322	329	322	336	1
323	328	322	329	329	357	349	344	336	336	1
336	317	327	324	327	322	323	328	338	335	1
107	90	60	86	101	125	170	188	69	102	1
71	141	206	80	67	70	58	58	144	77	1
77	63	71	76	64	113	67	90	88	78	1

copiando y pegando los datos. En la tabla 2.2 se presentan algunas entradas del dataset, cada columna representa la cantidad de mensajes contados en un intervalos de 5 segundos.

También se dispuso de un dataset externo llamado *dataset_sdn.csv* que es un conjunto de datos específico para SDN generado utilizando el emulador mininet y utilizado para la clasificación de tráfico por algoritmos de aprendizaje automático y aprendizaje profundo. Este dataset dispone de 23 características, entre ellas están la cantidad de *PACKET_IN* en intervalos de 30 segundos, por lo que es necesario formatear esta tabla para tomarlo por intervalos de 5 segundos.

2.6. Entrenamiento del modelo

El modelo de aprendizaje supervisado utilizado fue el de árbol de decisión. El modelo fue realizado utilizando la libreria **Scikit-learn**. El proceso de entrenamiento fue típico: Cargar dataset, segmentar características y etiquetas, entrenar y exportar modelo, testear el modelo. El 20 % del dataset se utilizó para el testeo y para determinar la precisión del modelo. El código es corto, por lo que se ha decidido presentarlo a continuación:

```
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# Leer el archivo de Excel
df = pd.read_excel('DataSetFinal.xls')
# Dividir el conjunto de datos en caracteristicas y etiquetas
X = df.iloc[:, :-1]
y = df.iloc[:, -1]    # La ultima columna es la etiqueta
X_train, X_test, y_train, y_test = train_test_split(X, y, \
test_size=0.2, random_state=42)
modelo = DecisionTreeClassifier()
modelo.fit(X_train, y_train)
# Exportar el modelo entrenado con pickle
with open('Modelo1.pkl', 'wb') as archivo:
    pickle.dump(modelo, archivo)
print(X_test)
# Cargar el modelo guardado con pickle
with open('Modelo1.pkl', 'rb') as archivo:
    modelo_cargado = pickle.load(archivo)
# Utilizar el modelo cargado
predicciones = modelo_cargado.predict(X_test)
presicion = accuracy_score(y_test,predicciones)
print("Presicion: ", presicion)
# Imprimir predicciones
print("Predicciones:", predicciones)
```

Capítulo 3

Analisis de resultados

3.1. Rendimiento de Docker

El contenedor Docker de RYU se limitó para dimensionar las características del controlador de acuerdo a la red propuesta. Al inicio de la ejecución el consumo de CPU oscila entre un 1.42 % y 3.4 % considerando que 20 de los 26 host se encuentran enviando tráfico del TCP y UDP preiodica y aleatoriamente. Esta información se obtiene ejecutando el comando **docker stats**. El consumo incrementa a un 23 % cuando se realiza un *pingall* en mininet para que los nodos conozcan sus rutas inicialmente (figura 3.1).

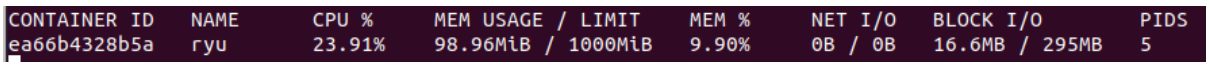


Figura 3.1: Consumo de CPU del controlador cuando se ejecuta un *pingall* en mininet.

Mientras que en la gráfica 3.2 se muestra la cantidad de mensajes *PACKET_IN* en la red cuando no existen ataques.

Figura 3.2: Mensajes *PACKET_IN* que llegan al controlador.

Los picos son los momentos en los que algunos host envían tráfico TCP o UDP a nuevas rutas, por lo que necesitan que el controlador les indique la trayectoria que deben seguir.

En la figura 3.1 también se puede apreciar el consumo de memoria en 98.96 Mb es decir un 9.98 % de la asignada al contenedor.

El consumo máximo presentado en la red mientras mantiene un tráfico normal fue de un 9.43 % de CPU y de 9.92 % de memoria, como se presenta en la siguiente imagen.

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT    MEM %     NET I/O     BLOCK I/O         PIDS
ea66b4328b5a   ryu       9.43%     99.18MiB / 1000MiB   9.92%     0B / 0B     16.6MB / 295MB    5
```

Figura 3.3: Consumo de CPU del controlador con tráfico normal.

Cuando los nodos de la red ya conocen las rutas para llegar a otros nodos, estos no necesitan enviar mensajes *PACKET_IN* al controlador y esto reduce el consumo de CPU y de memoria. Cabe resaltar que el modelo propuesto en este documento funciona a nivel de capa de control, es decir, cuando el controlador sufre ataques. En la figura 3.4 se muestra una gráfica del consumo de CPU vs el número de hosts atacantes.

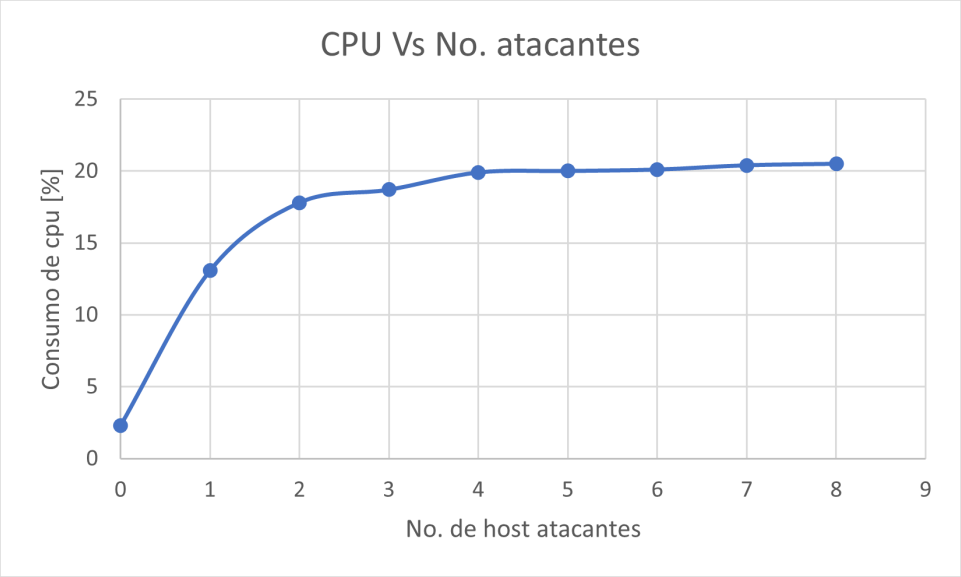


Figura 3.4: Consumo de CPU del controlador vs el número de host atacantes.

En la figura 3.4 se observa que el consumo de procesamiento cuando se realizan ataques desde 4 hosts es del 20 %, recordando que la cantidad de procesador asignado al contenedor fue del 0.2, tendríamos un controlador completamente saturado, es decir, el 20 % de consumo de CPU significa que el procesador del contenedor está procesando al 100 % de su capacidad y por ello colapsa y no tiene la capacidad de responder a nuevas solicitudes de trayectoria cono se demostró en la sección 2.

3.2. Precisión de los modelos generados

Como previamente se indicó, se trabajó con tres modelos generados a partir de datasets con diferente cantidad de entradas. El primer modelo se generó con 900 registros, el segundo tiene 2300 registros y el tercer contiene 10000 registros. Del dataset externo se han extraído 50000 registros, de la cantidad de mensajes *PACKET_IN* pero fue necesario formatear esta información ya que el tiempo en el que se cuentan los mensajes no es fija, por ejemplo hay duraciones de tiempo entre 100 y 1881 segundos, por lo que se segmentó cada entrada en intervalos de 5 segundos, dividiendo la cantidad de mensajes para cada intervalo.

El código generador de modelo realiza un primer análisis de la precisión del modelo, segmentando el dataset en 80 % para entrenamiento y 20 % para test, para

cada uno de los modelos los resultados que se obtuvieron se encuentran en la tabla 3.1, también se hace referencia en esta tabla a la cantidad de registros generados con la red para el entrenamiento del modelo.

Tabla 3.1: Precisión de los modelos determinado en Python con el set de prueba del dataset de entrenamiento.

Modelo	Precisión	Cantidad de registros
1	76 %	900
2	86 %	2300
3	94 %	10000

La precisión obtenida con cada uno de los modelos en Python es para la misma topología de red y en base a los scripts de tráfico asignados a los diferentes hosts definidos en la sección anterior, por lo que para evaluar la eficiencia y adaptabilidad de los tres modelos se realizaron pruebas en la misma topología, pero aumentando y disminuyendo la cantidad de nodos que envían tráfico.

EL modelo 1 que tiene unas cientos de entradas predice correctamente mientras la red no sea alterada, es decir, cuando en la red se tienen mas nodos generando trafico o buscando nuevas rutas, el modelo lo detecta como un ataque, esto se puede observar en la siguiente figura (3.5).

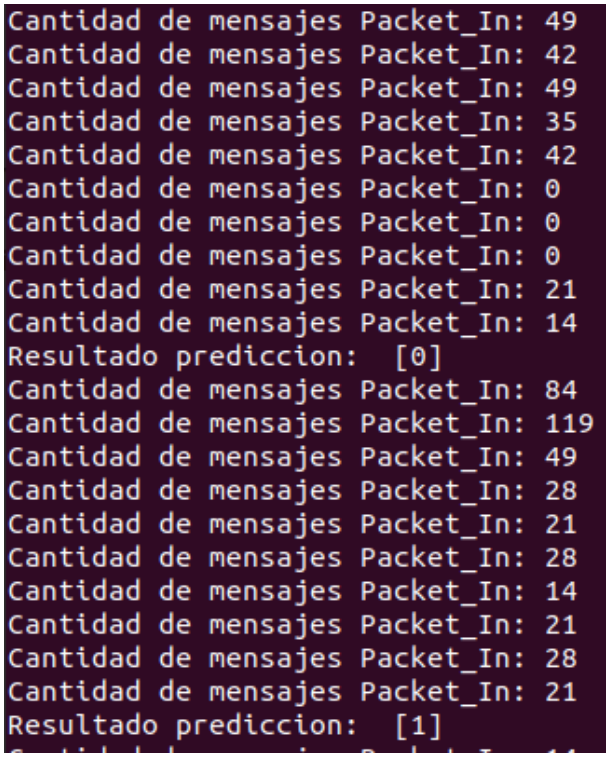


Figura 3.5: Detección de ataques del modelo 1.

El segundo modelo que fue entrenado con el dataset de 2300 registros propios es mas flexible que el anterior; este modelo no toma como un ataque cuando existe variación en el tráfico de la red, sin embargo, cuando el tráfico aumenta considerablemente ya indica que existe un ataque, ver figura 3.6

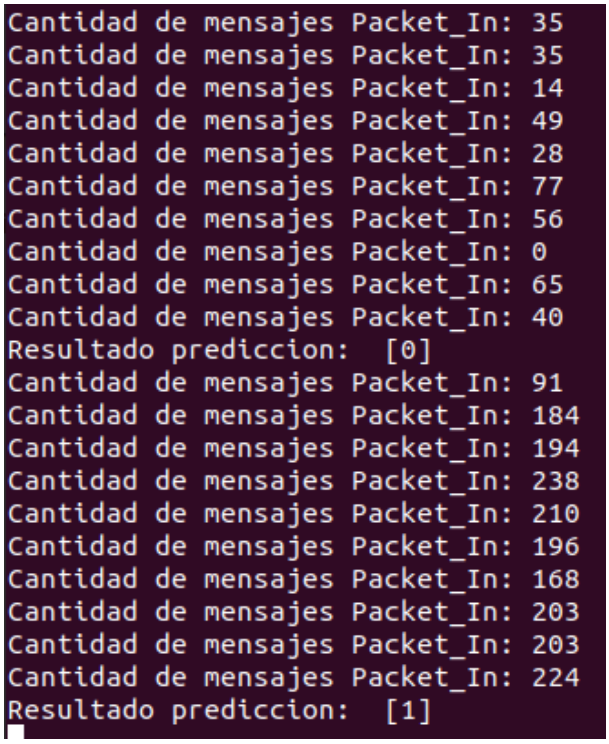


Figura 3.6: Detección de ataques del modelo 2.

El tercer modelo es mas robusto que los anteriores, la causa es porque al contener una mayor cantidad de registros con tráfico variable el modelo es mucho mas dinámico. Este modelo se probó enviando trafico desde nodos adicionales a los que ya estaban enviando tráfico, además se realizaron varios ataques desde un único host por cortos periodos de tiempo como se muuestra en la figura 3.7 y se etiquetó a este trafico como '0' en el dataset, es decir, como sin ataque, ya que el ataque desde un solo host no afecta mayormente al controlador, sin embargo al tráfico en donde se lanzaron ataques desde dos uno o mas nodos durante lapsos mas prolongados de tiempo se etiquetaron con '1' para que el modelo sepa cuando la red realmente está sufriendo un ataque que pueda alterar el correcto funcionamiento del controlador.

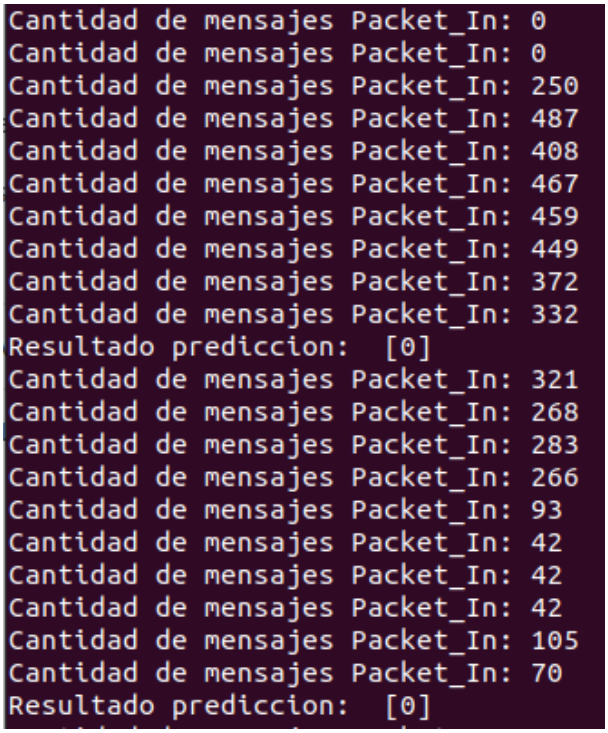


Figura 3.7: Detección de ataques del modelo 3, tráfico normal.

Si se comparan las figuras 3.7 y 3.8 no es fácilmente distinguible que tipo de tráfico es maligno o benigno, sin embargo la imagen 3.7 no contiene tráfico bajo ataque mientras que 3.8 si y se puede observar que el modelo detecta correctamente cada caso.

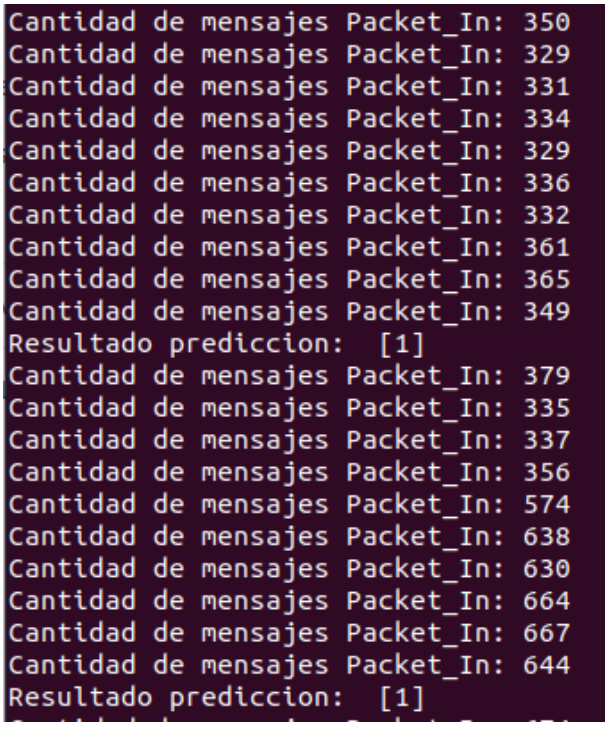


Figura 3.8: Detección de ataques del modelo 3, red bajo ataque.

Se realizaron capturas de 100 de las salidas de cada modelo para diferentes estados de la red para verificar si cada modelo detecta como verdadero positivo o negativo o como falso positivo o negativo, en la tabla 3.2 se muestran estos resultados en donde *verdadero positivo* indica que la red está siendo atacada y el modelo detecta el ataque , *falso positivo* indica que la red no está siendo atacada pero el modelo detecta un ataque, *verdadero negativo* implica que la red no está sufriendo ataques y el modelo no detecta ningún ataque y *falso negativo* indica que la red está siendo atacada pero el modelo no detecta el ataque. Se realizaron 50 ataques y en los otros 50 casos solo se envió trafico normal desde host adicionales.

Tabla 3.2: Resultados del modelo en términos de Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos, Falsos Negativos y Precisión.

Modelo	Verdadero Positivo	Verdadero Negativo	Falso Positivo	Falso Negativo	Precisión
1	46	30	4	20	0.76
2	50	36	0	14	0.86
3	50	44	0	6	0.94

Se muestran estos resultados de forma gráfica a continuación, en donde evidentemente, el tercer modelo generado tiene mayor precisión.

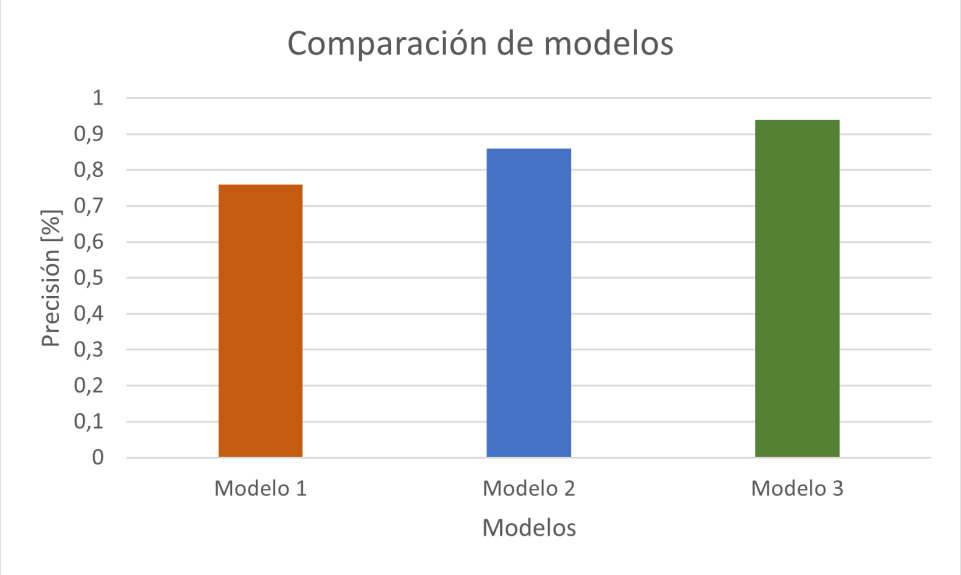


Figura 3.9: Gráfico estadístico de la precisión de los tres modelos generados.

Capítulo 4

Conclusiones y Trabajos Futuros

La detección de ataques utilizando inteligencia artificial ayuda a reconocer patrones de comportamiento de tráfico que no son fácilmente perceptibles para la mente humana, el administrador de la red. En el modelo final generado a partir de un dataset mas extenso se pudo observar una mayor eficiencia al hecho de que este contiene trafico mas dinámico, variable, por lo que la detección cubre un rango mas amplio de situaciones en las que se puede encontrar la red.

La cantidad de registros del dataset para le generación del modelo de detección es muy importante debido a que mientras mas información se tenga del comportamiento dinámico de la red, mas preciso es al momento de predecir un ataque y con ello evitar la detección de falsos positivos o falsos negativos.

Es conveniente desarrollar una topología con características cercanas a un entorno real, características como limitación de ancho de banda en los enlaces, latencias, tráfico de tiempo periódico y aleatorio, capacidades limitadas de los recursos del controlador, etc permiten que la detección de ataques se realice correctamente; una topología ideal en el simulador no permite obtener un aprendizaje correcto debido a que para la dimensión de la red de prueba, los recursos computacionales del equipo anfitrión sobrepasaban las requeridas.

El controlador RYU no presta una interfaz gráficas, lo que puede parecer tedioso al momento de administrar a diferencia de otros controladores de pago o incuso gratuitos como OpenDayLight, sin embargo RYU ha demostrado ser ligero y eficiente en la implementación de nuevas aplicaciones para la gestión de los nodos

de red, esta facilidad para la instalación y para el desarrollo de nueva aplicaciones basadas en python hacen de este controlador la mejor opción.

Gracias a la biblioteca de aprendizaje automático de python, SciKit-learn y al dataset generado mediante diversas pruebas se pudo obtener un modelo que cumple con los objetivos planteados, si bien aún quedan por estudiar algunos aspectos relacionados con el rendimiento en entornos reales, se considera que el modelo aquí generado tiene la capacidad de manejar correctamente ataques reales.

Para el correcto funcionamiento de la simulación se recomienda verificar las versiones de Python con las que trabaja mininet y el controlador RYU, y con la que se trabaja para el desarrollo de las aplicaciones, porque pese a que se puede trabajar con las mismas librerías, las nuevas versiones tienen pequeñas actualizaciones que generan errores al momento de ejecutar los códigos.

Se recomienda la utilización de Docker para dimensionar las capacidades del controlador de acuerdo con la dimensión de la red. La red propuesta en este trabajo es relativamente pequeña, por lo que si el controlador utiliza toda la capacidad computacional del equipo anfitrión, incluso lanzando ataques desde todos los nodos, el controlador no se vería afectado. Se debe considerar que en una red real se llegan a tener de cientos a miles de nodos que generan tráfico.

Considerando la complejidad y la naturaleza dinámica de los ataques cibernéticos, una dirección prometedora sería la exploración de modelos de aprendizaje profundo, como por ejemplo redes neuronales convolucionales (CNN) o redes neuronales recurrentes (RNN), para mejorar aún más la precisión de detección.

Una etapa crítica del trabajo futuro podría ser la implementación y evaluación del modelo en un entorno de producción real, para medir su eficacia y aplicabilidad en condiciones del mundo real además de la implementación de un sistema de mitigación de ataques que tome las medidas preventivas y correctivas para tratar con la amenaza.

Glosario

docker plataforma de código abierto para el desarrollo, empaquetado y ejecución de aplicaciones en entornos llamados "contenedores".

DoS Denial of service (denegación de servicios).

HPING3 Herramienta de linea de comandos para prueba de penetración y análisis de redes..

iperf Herramienta de línea de comandos, para medir el rendimiento de la red y generar de tráfico.

PING Comando utilizado en redes de computadoras para verificar conectividad entre dispositivos.

RYU Controlador para redes definidas por software.

scapy herramienta de manipulación y generación de paquetes de red en Python.

SDN Redes definidas por software.

TCP Protocolo de control transmisión.

UDP Protocolo de datagramas de usuario.

VM Maquina virtual.

Referencias

[1] R. Deb y S. Roy, «A comprehensive survey of vulnerability and information security in SDN,» *Computer Networks*, vol. 206, abr. de 2022, ISSN: 13891286. DOI: 10.1016/j.comnet.2022.108802.

[2] J. Correa, J. Imbachi y J. Botero, «Security in SDN: A comprehensive survey,» *Journal of Network and Computer Applications*, vol. 159, jun. de 2020, ISSN: 10958592. DOI: 10.1016/j.jnca.2020.102595.

[3] Y. Jarraya, T. Madi y M. Debbabi, «A survey and a layered taxonomy of software-defined networking,» *IEEE Communications Surveys and Tutorials*, vol. 16, págs. 1955-1980, 4 abr. de 2014, ISSN: 1553877X. DOI: 10.1109/COMST.2014.2320094.

[4] B. Nunes, M. Mendonca, N. Nguyen, K. Obraczka y T. Turletti, «A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,» *HAL open science*, vol. 16, págs. 1617-1634, 3 2014. DOI: 10.1109/SURV.2014.012214.00180Ãŕ. dirección: https://inria.hal.science/hal-00825087v5.

[5] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky y S. Uhlig, «Software-defined networking: A comprehensive survey,» *Proceedings of the IEEE*, vol. 103, págs. 14-76, 1 ene. de 2015, ISSN: 15582256. DOI: 10.1109/JPROC.2014.2371999.

[6] N. Ahuja, G. Singal y D. Mukhopadhyay, «DDOS attack SDN Dataset,» vol. 1, 2020. DOI: 10.17632/JXPFJC64KR.1.

[7] M. Akbanov, V. G. Vassilakis y M. D. Logothetis, «WannaCry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms,» *Journal of Telecommunications and Information Technology*, págs. 113-124, 1 2019, ISSN: 18998852. DOI: 10.26636/jtit.2019.130218.

[8] Y. Gu, A. Mccallum y D. Towsley, «Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation,» University of Massachusetts.

[9] M. McNickle, *Cinco protocolos SDN que no son OpenFlow | Computer Weekly*, sep. de 2014. dirección: https://www.computerweekly.com/es/cronica/Cinco-protocolos-SDN-que-no-son-OpenFlow.

[10] D. Briain, «RYU SDN [] Testbed Manual,» Institute of technology Carlow, mayo de 2020.

[11] J. Henao, «GUÍA DE IMPLEMENTACIÓN Y USO DEL EMULADOR DE REDES MININET,» 2015.

[12] 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, págs. 1797-1805, 3 jun. de 2023, ISSN: 23029285. DOI: 10.11591/eei.v12i3.5232.

[13] M. Pietro, *Mininet Lab 1: Introduction to Mininet - HackMD*, oct. de 2019. dirección: https://hackmd.io/@pmanzoni/BklqpKddS.

[14] L. Bob, H. Nikhil, H. Brandon y J. Vimal, *Introduction to Mininet · mininet/mininet Wiki · GitHub*, mar. de 2021. dirección: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.

[15] S. Wang, J. F. Balarezo, K. G. Chavez et al., «Detecting flooding DDoS attacks in software defined networks using supervised learning techniques,» *Engineering Science and Technology, an International Journal*, vol. 35, nov. de 2022, ISSN: 22150986. DOI: 10.1016/j.jestch.2022.101176.

[16] D. Tang, X. Wang, Y. Yan, D. Zhang y H. Zhao, «ADMS: An online attack detection and mitigation system for LDoS attacks via SDN,» *Computer Communications*, vol. 181, págs. 454-471, ene. de 2022, ISSN: 1873703X. DOI: 10.1016/j.comcom.2021.10.007.

[17] M. S. Tok y M. Demirci, «Security analysis of SDN controller-based DHCP services and attack mitigation with DHCPguard,» *Computers and Security*, vol. 109, oct. de 2021, ISSN: 01674048. DOI: 10.1016/j.cose.2021.102394.

[18] A. Javed y L. Seemab, «Handling Intrusion and DDoS Attacks in Software Defined Networks Using Machine Learning Techniques,» *2014 National Software Engineering Conference (NSEC) 11-12 Nov. 2014, Rawalpindi, Pakistan*, 2014.

[19] J. G. Hsieh, Y. L. Lin y J. H. Jeng, «Preliminary study on Wilcoxon learning machines,» *IEEE Transactions on Neural Networks*, vol. 19, págs. 201-211, 2 feb. de 2008, ISSN: 10459227. DOI: 10.1109/TNN.2007.904035.

[20] S. Mukkamala, G. Janoski y A. Sung, «Intrusion Detection Using Neural Networks and Support Vector Machines,» New Mexico Institute of Mining y Technology.

[21] D. Comaneci y C. Dobre, «Securing networks using SDN and machine learning,» Institute of Electrical y Electronics Engineers Inc., dic. de 2018, págs. 194-200, ISBN: 9781538676486. DOI: 10.1109/CSE.2018.00034.

[22] M. Nisharani, D Narayan y P Baligar, «Detection of Distributed Denial of Service Attacksusing Machine Learning Algorithms in SoftwareDefined Networks,» *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI) : 13-16 Sept. 2017.*, sep. de 2017.

[23] B. Sarma, R. Kumar y T. Tuithung, «Machine learning enabled network and task management in SDN based Fog architecture,» *Computers and Electrical Engineering*, vol. 108, mayo de 2023, ISSN: 00457906. DOI: 10.1016/j.compeleceng.2023.108705.