

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERIA EN CIENCIAS APLICADAS

CARRERA DE TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"DETECCIÓN DE ATAQUES DDOS DE BAJA INTENSIDAD EN SDN MEDIANTE ANÁLISIS DE ENTROPÍA DE FLUJOS Y MODELOS DE MACHINE LEARNING INTERPRETABLES"

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

Línea de investigación: Desarrollo, aplicación de software y cibersecurity (seguridad cibernética)

AUTOR:

Nahim Jhosue Gómez González

DIRECTOR:

Ing. Fabián Geovanny Cuzme Rodríguez, Msc

Ibarra - Ecuador, 2026

UNIVERSIDAD TÉCNICA DEL NORTE
DIRECCIÓN DE BIBLIOTECA

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	
APELLIDOS Y NOMBRES:	GÓMEZ GONZÁLEZ NAHIM JHOSUE

DATOS DE LA OBRA	
TÍTULO:	"DETECCIÓN DE ATAQUES DDOS DE BAJA INTENSIDAD EN SDN MEDIANTE ANÁLISIS DE ENTROPÍA DE FLUJOS Y MODELOS DE MACHINE LEARNING INTERPRETABLES"
AUTOR (ES):	NAHIM JHOSUE GÓMEZ GONZÁLEZ
FECHA: DD/MM/AAAA	28/07/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR /ASESOR:	ING. FABIÁN GEOVANNY CUZME RODRÍGUEZ, MSC ING. MAYA OLALLA EDGAR ALBERTO, MSC

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 la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 28. días del mes de julio de 2026

EL AUTOR:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
NAHIM JHOSUE GOMEZ GONZALEZ

.....

Nahim Jhosue Gómez González

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 28 de Julio de 2026

ING. FABIÁN GEOVANNY CUZME RODRÍGUEZ, MSC
DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
FABIAN GEOVANNY CUZME RODRIGUEZ

……

Ing. Fabián Geovánny Cuzme Rodríguez, Msc
C.C.: 1311527012

DEDICATORIA

Dedico este trabajo a quienes estuvieron conmigo desde el inicio de este camino, a quienes me brindaron su apoyo y supieron tenderme la mano en los momentos difíciles.

A mis docentes, quienes vieron en mí un potencial y confiaron en él cuando más lo necesitaba.

A mi familia, mis amigos y mi pareja, por su compañía incondicional; y a mi tío, quien contribuyó a formarme como una persona responsable, ética y con valores.

De manera especial, a mis padres, que me acompañan en cada paso, y a mi hermano, quien ya no está presente, pero cuya memoria vive en cada uno de mis logros.

Nahim Jhosue Gómez González

AGRADECIMIENTO

A Dios, por derramar sus bendiciones sobre mi vida, darme la fortaleza necesaria en cada etapa y permitirme alcanzar este logro tan importante.

A mis padres, quienes me inculcaron buenos valores y me brindaron su apoyo incondicional a lo largo de toda mi vida.

A la Universidad Técnica del Norte, por abrirme sus puertas y, a través de sus docentes, brindarme los conocimientos y experiencias que hicieron posible mi formación profesional.

Al Ingeniero Fabián Geovanny Cuzme Rodríguez, director de este trabajo, por su tiempo, sus valiosas lecciones y su acertada guía durante todo el desarrollo del proyecto.

Al Ingeniero Edgar Alberto Maya Olalla, por su tiempo y dedicación en la asesoría de este proyecto, aportes que fueron fundamentales para su culminación.

Al Ingeniero Jaime Roberto Michilena Calderón, por su orientación y sus sabios consejos frente a las adversidades que se presentaron en el transcurso de este trabajo.

Finalmente, a César y Keneth, quienes siempre confiaron en mí y en todo lo que soy capaz de lograr.

Nahim Jhosue Gómez González

RESUMEN

El presente trabajo investigativo aborda los ataques DDoS de baja intensidad en redes definidas por software, los cuales degradan progresivamente la disponibilidad de los servicios sin generar picos evidentes de tráfico, dificultando su detección mediante mecanismos tradicionales basados en umbrales o volumen de paquetes. El propósito global consistió en la implementación de un sistema de detección de ataques DDoS low-rate en redes SDN a partir del análisis de la entropía de flujos y de modelos de Machine Learning interpretables. Para ello, se hizo uso de una metodología basada en el modelo en cascada que contempló el análisis del estado del arte, la definición de requerimientos a partir de la norma ISO/IEC/IEEE 29148, el diseño de la arquitectura SDN, la implementación (Ryu, OpenFlow, Entropía de Rényi y XGBoost) y la evaluación en tres escenarios de prueba. Además, se empleó STRIDE para analizar amenazas, con énfasis en denegación de servicio. Los resultados mostraron que sin seguridad los ataques ICMP, TCP y UDP afectaron la estabilidad de la red, elevaron la carga del controlador y degradaron servicios HTTP y VoIP. Con seguridad el sistema alcanzó 98,95 % de exactitud en ICMP mixto y 99,51 % en TCP mixto, aunque presentó limitaciones en UDP/VoIP por falsos positivos. A partir de los resultados obtenidos, se evidencia que el sistema propuesto contribuye al fortalecimiento de la seguridad en redes SDN al permitir la detección, análisis y respuesta frente a ataques DDoS de baja intensidad, integrando métricas de comportamiento del tráfico, clasificación automática e interpretabilidad para mejorar la visibilidad del plano de control y apoyar la toma de decisiones ante eventos que afectan la disponibilidad de la red.

Palabras clave: SDN, DDoS de baja intensidad, entropía de Rényi, XGBoost, SHAP, Ryu, OpenFlow, STRIDE.

ABSTRACT

This research addresses low-rate Distributed Denial-of-Service (DDoS) attacks in Software-Defined Networks (SDNs), which progressively degrade service availability without generating evident traffic spikes, thereby making their detection difficult through traditional mechanisms based on thresholds or packet volume. The main objective was to implement a low-rate DDoS attack detection system for SDN environments based on flow entropy analysis and interpretable Machine Learning models. To achieve this objective, a waterfall model methodology was adopted, encompassing a state-of-the-art review, requirements definition based on the ISO/IEC/IEEE 29148 standard, SDN architecture design, system implementation using Ryu, OpenFlow, Rényi entropy, and XGBoost, and evaluation across three test scenarios. Additionally, the STRIDE framework was applied to analyze security threats, with particular emphasis on denial-of-service attacks. The results showed that, in the absence of security mechanisms, ICMP, TCP, and UDP attacks affected network stability, increased controller workload, and degraded HTTP and VoIP services. With the security system enabled, the proposed solution achieved an accuracy of 98.95% for mixed ICMP traffic and 99.51% for mixed TCP traffic, although limitations were observed in UDP/VoIP traffic due to false positives. Based on the results obtained, the proposed system contributes to strengthening security in SDN environments by enabling the detection, analysis, and mitigation of low-rate DDoS attacks. It integrates traffic behavior metrics, automated classification, and model interpretability to improve control-plane visibility and support decision-making in response to events that affect network availability.

Keywords: SDN, low-rate DDoS, Rényi entropy, XGBoost, SHAP, Ryu, OpenFlow, STRIDE

Tabla de contenido

DEDICATORIA iii

AGRADECIMIENTO iv

RESUMEN v

ABSTRACT vi

Capítulo I: Antecedentes 1

1.1 Tema 1

1.2 Problema 1

1.3 Objetivos 3

1.3.1 Objetivo General 3

1.3.2 Objetivos Específicos 3

1.4 Alcance 4

1.5 Justificación 9

Capítulo II: Fundamentación Teórica 11

2.1 Ciberseguridad 11

2.1.1 Seguridad de la información 12

2.1.2 Modelo CIA 12

2.1.3 Ataques 14

2.1.3.1 Ataques DoS. 15

2.1.3.2 Ataques DDoS.. 16

2.2 Redes Definidas por Software (SDN) 16
2.2.1 Arquitectura SDN 16
2.2.2 Protocolo OpenFlow 19
2.2.2.1 Mensajes OpenFlow.. 21
2.3 Seguridad en SDN 22
2.3.1 Vulnerabilidades en SDN 22
2.3.1.1 Desbordamiento de la tabla de flujo.. 23
2.3.1.2 Congestión del enlace de control y datos 23
2.3.1.3 Saturación del controlador.. 23
2.3.2 Ataques DDoS de baja intensidad en SDN 23
2.3.3 Sistemas IDS en SDN 24
2.4 Modelos ML 25
2.4.1 Modelos Machine learning Explicables 26
2.5 Teoría de la Entropía 27
2.5.1 Entropía de Shannon 28
2.5.2 Entropía de Rényi 28
2.5.3 Entropía aplicada a redes SDN 29
2.5.4 Métricas de evaluación 31
2.5.4.1 Precisión.. 31
2.5.4.2 Recall (Sensibilidad).. 32

2.5.4.3 F1-Score. 32

2.5.4.4 Exactitud. 32

Capítulo III: Despliegue y Funcionamiento 33

3.1 Metodología 33

3.2 Análisis 34

3.2.1 Análisis de trabajos relacionados 35

3.2.2 Análisis de la situación actual 36

3.2.3 Requerimientos 38

3.2.3.1 Requerimientos de Stakerholders (StSR) 39

3.2.3.2 Requerimientos del Sistema (SySR). 40

3.2.3.3 Requerimientos de Software (SRS). 42

3.2.3.4 Requerimientos de Hardware (SRH). 43

3.2.3.5 Especificación de Requerimientos del Modelo (MRS). 44

3.2.3.6 Especificación de Requerimientos de Arquitectura (SyRA). 45

3.2.4 Selección de Software 46

3.2.4.1 Selección del Software de Simulación. 47

3.2.4.2 Selección del Modelo 48

3.2.4.3 Selección del Modelo Interpretable. 50

3.2.4.4 Selección del Controlador 51

3.2.4.5 52

3.2.4.6 Selección del Entorno de Programación. 53
3.3 Diseño 54
3.3.1 Diseño de la topología 55
3.3.1.1 Dispositivos que conforman la Red.. 59
3.3.1.2 Direccionamiento IPv4 de la topología..... 60
3.3.2 Diseño de la arquitectura del controlador y monitoreo..... 61
3.3.2.1 Arquitectura del controlador Ryu.. 61
3.3.2.2 Diseño del Dashboard de visualización del controlador.. 65
3.3.3 Diseño del modelo de detección 71
3.3.3.1 Diagrama de Bloques del Modelo..... 71
Bloque de Generación de Datos (Entorno SDN). 72
Bloque de Preprocesamiento de Datos. 73
Bloque de Unidad de Entrenamiento XGBoost. 73
3.3.3.2 Diagrama de Secuencia del Sistema de Detección.. 74
3.3.4 Diseño del pipeline de datos 76
3.3.4.1 Origen de datos del Dataset. P..... 76
3.3.4.2 Diagramas de Flujo 77
Flujograma de adquisición y carga de datos. 77
Flujograma de Preprocesamiento de Datos..... 78
Flujograma del Entrenamiento y Evaluación de Modelo. 81

3.3.4.3 Técnicas de Preprocesamiento y entrenamiento utilizadas.. 83

Estrategia de preprocesamiento de datos. 83

Estrategia de entrenamiento del modelo. 84

3.3.5 Diseño del sistema de ataques 85

3.3.5.1 Taxonomía de ataques.. 85

3.3.5.2 Diseño del Dashboard de administración de atacantes. E 86

3.3.6 Diseño del sistema del modelo interpretable SHAPASH 92

3.3.6.1 Cabecera del sistema.. 93

3.3.6.2 Importancia global de características.. 93

3.3.6.3 Exploración del conjunto de datos.. 93

3.3.6.4 Contribución de características. 93

3.3.6.5 Explicación local.. 93

3.3.7 Tabla resumen de librerías pensadas a implementar 93

3.4 Implementación 96

3.4.1 Implementación de la topología 96

3.4.1.1 Despliegue de dispositivos y enlaces de la red SDN. 97

3.4.1.2 Implementación de la topología para el conjunto de datos con Mininet 99

3.4.1.3 Configuración del direccionamiento IPv4 y conectividad del conjunto de datos. 100

3.4.2 Implementación de la arquitectura del controlador y monitoreo 100

3.4.2.1 Implementación de la aplicación del controlador Ryu..... 101

3.4.2.2 Gestión de flujos y eventos Packet-In..... 102

3.4.2.3 Implementación del Dashboard de visualización del controlador. 103

3.4.2.4 Implementación de la aplicación del controlador Ryu..... 107

3.4.3 Implementación del modelo de detección..... 108

3.4.3.1 Integración del cálculo de entropía en el sistema. 109

3.4.3.2 Implementación del modelo XGBoost.....110

3.4.3.3 Implementación del flujo de inferencia en tiempo real. 110

3.4.3.4 Implementación de las acciones de mitigación.....111

3.4.4 Implementación del pipeline de datos.....111

3.4.4.1 Implementación del origen y adquisición de datos.....112

3.4.4.2 Implementación del preprocesamiento de datos.. 115

Limpieza de datos y depuración de valores. 115

Ingeniería de características: Entropía de Rényi.....116

Normalización y transformación de datos. 117

Balanceo con SMOTE y UnderSampler. 118

3.4.4.3 Implementación del entrenamiento del modelo..... 120

División de los datos para entrenamiento, validación y prueba..... 120

Entrenamiento del modelo XGBoost. 121

Estimación del rendimiento del Modelo XGBoost. 123

Matriz de Confusión del modelo XGBoost. 124

Auc ROC y Loss Funnction del modelo XGBoost. 125

Exportación del modelo para su integración con Ryu. 128

3.4.4.4 Implementación del Dashboard de administración de atacantes 130

3.4.5 Implementación del sistema del modelo interpretable SHAPASH..... 139

3.4.5.1 Importación de bibliotecas. 139

3.4.5.2 Definición de rutas.. 140

3.4.5.3 Carga del modelo entrenado.. 140

3.4.5.4 Carga del escalador.. 140

3.4.5.5 Recuperación de las características. 141

3.4.5.6 Carga del conjunto de datos 141

3.4.5.7 Validación de la estructura de los datos.. 141

3.4.5.8 Separación de variables predictoras y variable objetivo. 142

3.4.5.9 Conversión de etiquetas a valores numéricos. 142

3.4.5.10 Escalamiento de las características.. 144

3.4.5.11 Definición del diccionario de clases 144

3.4.5.12 Creación del explicador SHAPASH.. 145

3.4.5.13 Compilación del explicador. 145

Capítulo IV: Evaluación de Resultados 148

4.1 Plan de ataques 148

4.1.1 Criterios considerados para el desarrollo del plan 149

4.1.2 Nomenclatura de los casos de prueba 150

4.1.3 Plan general de ataques por escenarios 152

4.1.4 Descripción operativa de los escenarios 160

4.1.4.1 Primer escenario: red sin sistema de seguridad. 162

Verificación de la topología. 162

Verificación del direccionamiento IP. 164

Usuarios normales 164

Usuarios atacantes. 164

Servidor web. 165

Controlador Ryu. 166

Verificación del controlador Ryu. 166

Verificación de los switches Open vSwitch. 167

Prueba ICMP con usuarios normales. 168

Prueba ICMP únicamente con atacantes 172

Prueba ICMP mixta 174

Prueba TCP con usuarios normales 177

Prueba TCP con atacantes 182

Prueba TCP mixta 184

Prueba UDP con llamada normal 187

Prueba UDP con llamada bajo ataque 189

4.1.4.2 Resultados 192

4.1.4.3 Segundo escenario: red con sistema de seguridad 192

4.1.4.4 Tercer escenario: sistema de seguridad externo 197

4.2 Análisis de los resultados con el método STRIDE 203

4.2.1.1 Análisis de la denegación de servicio por escenario 207

4.2.1.2 Valoración general mediante STRIDE 207

Conclusiones 209

Recomendaciones 212

Bibliografía 215

Lista de Tablas

Tabla 1 14

Tabla 2 *Tipos de mensajes OpenFlow* 21

Tabla 3 *Modelos Machine Learning* 26

Tabla 4 *Modelos de aprendizaje automático explicables* 27

Tabla 5 *Estudios realizados con técnicas de entropía en SDN y Métricas Obtenidas.* 30

Tabla 6 *Requerimientos de Stakeholders* 40

Tabla 7 *Requerimientos del Sistema (SySR)* 40

Tabla 8 *Requerimientos de Software (SRS)* 43

Tabla 9 *Requerimientos de Hardware (SRH)* 44

Tabla 10 *Requerimientos del modelo* 45

Tabla 11 *Requerimientos de Arquitectura* 46

Tabla 12 *Selección del Software de Simulación* 47

Tabla 13 *Selección del Modelo* 49

Tabla 14 *Selección del Modelo Interpretable* 50

Tabla 15 *Selección del Controlador* 51

Tabla 16 *Selección del Lenguaje de Programación* 52

Tabla 17 *Selección del Entorno de Programación* 54

Tabla 18 *Cumplimiento de Requerimientos de arquitectura* 56

Tabla 19 *Dispositivos que conforman la topología.* 59

Tabla 20 *Direccionamiento IPv4 de la topología de red* 60

Tabla 21 *Librerías empleadas* 94

Tabla 22 *Comandos que permiten la asociación de los switches SDN al controlador* 98

Tabla 23 *Criterios considerados para el desarrollo del plan de ataques* 149

Tabla 24 *Nomenclatura de los casos de prueba* 150

Tabla 25 *Plan general de ataques por escenarios* 152

Tabla 26 *Descripción operativa de los escenarios* 160

Tabla 27 *Resultados obtenidos de la prueba de segundo escenario: red con sistema de seguridad* 193

Tabla 28 *Resultados obtenidos de la prueba de tercer escenario: red con sistema de seguridad externo* 197

Tabla 29 *Comparación de MTTD entre el escenario 2 y 3* 201

Tabla 30 *Análisis de los resultados de las pruebas de los diferentes escenarios con el método STRIDE* 204

Lista de Figuras

Figura 1 *Árbol de problemas ataques DDoS de Baja intensidad* 3

Figura 2 *Arquitectura correspondiente al escenario de evaluación del sistema de detección.* 7

Figura 3 *Triangulo C.I.A* 13

Figura 4 *Arquitectura de red basada en SDN* 17

Figura 5 *Arquitectura de red basada en SDN* 19

Figura 6 *Modelo en Cascada* 34

Figura 7 *Diseño de red propuesta para la Cooperativa de Segmento 1* 37

Figura 8 *Topología de red SDN* 58

Figura 9 *Arquitectura de un Controlador SDN* 62

Figura 10 *Arquitectura del Controlador Ryu* 64

Figura 11 *Diseño del Dashboard del controlador - primera pestaña* 66

Figura 12 *Diseño del Dashboard del controlador – pestaña Indicadores técnicos* 67

Figura 13 *Diseño del Dashboard del controlador - pestaña Topología de la red* 69

Figura 14 *Diseño del Dashboard del controlador - pestaña Registros* 70

Figura 15 *Diagrama de Bloques del Modelo* 72

Figura 16 *Diagrama de Secuencia del Sistema de Detección* 74

Figura 17 *Arquitectura del banco de pruebas* 76

Figura 18 *Flujograma de adquisición y carga de datos* 77

Figura 19 *Flujograma de Preprocesamiento de datos* 80

Figura 20 *Flujograma de entrenamiento y evaluación del modelo* 82

Figura 21 *Taxonomía de ataques DDoS clasificado por Volumétricos y por Baja intensidad* 86

Figura 22 *Diseño del Dashboard de administración de ataques - primera pestaña* 87

Figura 23 *Diseño del Dashboard de administración de ataques - segunda pestaña* 90

Figura 24 *Diseño del Dashboard del sistema del modelo interpretable SHAPASH* 92

Figura 25 *Establecimiento de los enlaces de la red SDN* 97

Figura 26 *Configuraciones de los enlaces OVS al controlador* 98

Figura 27 *Implementación de la estructura base del entorno SDN* 99

Figura 28 *Configuración del direccionamiento IPv4 y conectividad del conjunto de datos* 100

Figura 29 *Implementación de la aplicación del controlador Ryu* 102

Figura 30 *Gestión de flujos* 103

Figura 31 *Gestión de eventos Packet-In* 103

Figura 32 *Implementación del Dashboard de visualización del controlador* 104

Figura 33 *Implementación del servidor HTTP auxiliar* 104

Figura 34 *Dashboard implementado del controlador - primera pestaña* 105

Figura 35 *Dashboard implementado del controlador - pestaña Indicadores técnicos* 105

Figura 36 *Dashboard implementado del controlador - pestaña sobre la topología de la red* 106

Figura 37 *Dashboard del controlador - pestaña Registros* 107

Figura 38 *Dashboard implementado de administración de ataques - primera pestaña* 107

Figura 39 *Dashboard implementado de administración de ataques - segunda pestaña* 108

Figura 40 *Implementación del cálculo de entropía al sistema en el modelo de detección* 109

Figura 41 *Implementación del modelo XGBoost en el modelo de detección* 110

Figura 42 *Implementación del flujo de inferencia en tiempo real en el modelo de detección* 111

Figura 43 *Implementación de las acciones de mitigación en el modelo de detección* 111

Figura 44 *Implementación del origen y adquisición de datos* 112

Figura 45 *Carga del Dataset y categorías Generadas* 113

Figura 46 *Exploración y distribución del Dataset* 114

Figura 47 *Implementación de la limpieza de datos y depuración de valores* 115

Figura 48 *Implementación de la entropía de Rényi* 116

Figura 49 *Implementación de la Entropía de Rényi mediante la integración con el vector de características* 117

Figura 50 *Implementación de la normalización y transformación de datos* 118

Figura 51 *Implementación del balanceo con SMOTE y UnderSampler* 119

Figura 52 *Visualización del Balanceo de Clases* 120

Figura 53 *Implementación de la división de datos para entrenamiento, validación y prueba* ... 121

Figura 54 *Implementación del entrenamiento del modelo XGBoost* 122

Figura 55 *Generación de Gráficas del modelo* 123

Figura 56 *Resultado de las predicciones del Modelo en el conjunto de prueba* 124

Figura 57 *Matriz de confusión con los datos de test* 125

Figura 58 *Curva logarítmica Multiclase* 126

Figura 59 *Área bajo la curva ROC (AUC) del rendimiento del modelo* 127

Figura 60 *Implementación de la exportación del modelo* 129

Figura 61 *Implementación de la integración con Ryu* 129

Figura 62 *Implementación del Dashboard de administración de atacantes - verificación del motor Docker* 130

Figura 63 *Implementación del Dashboard de administración de atacantes - almacenamiento de bots y contenedores* 131

Figura 64 *Implementación del Dashboard de administración de atacantes - definición de tipos de ataques disponibles* 132

Figura 65 *Implementación del Dashboard de administración de atacantes - API de registro* ... 133

Figura 66 *Implementación del Dashboard de administración de atacantes - registro de la acción en historial y actualización de estado de los bots* 134

Figura 67 *Implementación del Dashboard de administración de atacantes - acción de bots de consulta al endpoint* 135

Figura 68 *Implementación del Dashboard de administración de atacantes - despliegue dinámico de los atacantes* 136

Figura 69 *Implementación del Dashboard de administración de atacantes - función de detención general* 137

Figura 70 *Dashboard de administración de atacantes - primera pestaña* 138

Figura 71 *Dashboard de administración de atacantes - segunda pestaña* 138

Figura 72 *Dashboard de administración de atacantes - segunda parte de la segunda pestaña* 139

Figura 73 *Implementación del sistema del modelo SHAPASH - importación de bibliotecas* 139

Figura 74 *Implementación del sistema del modelo SHAPASH - definición de rutas* 140

Figura 75 *Implementación del sistema del modelo SHAPASH – carga del modelo entrenado* .. 140

Figura 76 *Implementación del sistema del modelo SHAPASH - carga del escalador* 140

Figura 77 *Implementación del sistema del modelo SHAPASH - recuperación de las características* 141

Figura 78 *Implementación del sistema del modelo SHAPASH - carga del conjunto de datos* ... 141

Figura 79 *Implementación del sistema del modelo SHAPASH - validación de la estructura de los datos* 141

Figura 80 *Implementación del sistema del modelo SHAPASH - separación ded variables predictoras y variable objetivo* 142

Figura 81 *Implementación del sistema del modelo SHAPASH - conversión de etiquetas a valores numéricos* 142
Figura 82 *Implementación del sistema del modelo SHAPASH - comprobación de clases* 143
Figura 83 *Implementación del sistema del modelo SHAPASH - conversión de las etiquetas en valores enteros y serie de datos* 143
Figura 84 *Implementación del sistema del modelo SHAPASH - escalamiento de las características* 144
Figura 85 *Implementación del sistema del modelo SHAPASH - conversión de los datos escalados a dataframe* 144
Figura 86 *Implementación del sistema del modelo SHAPASH - definición de diccionario de clases* 144
Figura 87 *Implementación del sistema del modelo SHAPASH - creación del explicador* 145
Figura 88 *Implementación del sistema del modelo SHAPASH - compilación del explicador* 145
Figura 89 *Compilación del explicador - importación global de las características* 145
Figura 90 *Compilación del explicador - comparación de la matriz de confusión entre las clases reales con las clases predichas* 146
Figura 91 *Compilación del explicador - explicación de cada característica con respecto al resultado obtenido*..... 146
Figura 92 *Compilación del explicador - relación entre las características con respecto a su contribución en las predicciones realizadas por el modelo*..... 146
Figura 93 *Dashboard final del modelo interpretativo SHAPASH*..... 147
Figura 94 *Topología - Red sin sistema de seguridad*..... 162
Figura 95 *Verificación del direccionamiento IP – Usuarios normales*..... 164

Figura 96 *Verificación del direccionamiento IP – Usuarios atacantes* 164

Figura 97 *Establecimiento del servidor VoIP* 165

Figura 98 *Establecimiento del servidor web* 165

Figura 99 *Establecimiento del controlador Ryu* 166

Figura 100 *Verificación del controlador Ryu* 166

Figura 101 *Verificación de los switches Open vSwitch* 167

Figura 102 *Configuración y ejecución ICMP de los usuarios normales* 168

Figura 103 *Generación de solicitudes ICMP por parte de los usuarios normales establecidos* 170

Figura 104 *Visualización de las solicitudes ICMP Echo Request y respuestas Echo Reply* 170

Figura 105 *Proceso ryu-manager ICMP por parte de los usuarios normales* 171

Figura 106 *Configuración y ejecución ICMP únicamente con atacantes* 172

Figura 107 *Generación de solicitudes ICMP por parte de los atacantes* 173

Figura 108 *Proceso ryu-manager ICMP por parte únicamente de los atacantes* 174

Figura 109 *Configuración y ejecución ICMP mixta* 175

Figura 110 *Generación de solicitudes ICMP mixta* 175

Figura 111 *Generación de solicitudes ICMP observados* 176

Figura 112 *Proceso ryu-manager ICMP mixta* 177

Figura 113 *Configuración y ejecución TCP con usuarios normales* 178

Figura 114 *Generación de solicitudes TCP con usuarios normales* 179

Figura 115 *Captura de respuestas HTTP de las solicitudes TCP con usuarios normales* 180

Figura 116 *Retransmisiones TCP con usuarios normales* 181

Figura 117 *Paquetes TCP con usuarios normales* 181

Figura 118 *Proceso ryu-manager TCP con usuarios normales* 182

Figura 119 *Configuración y ejecución TCP con atacantes* 183

Figura 120 *Generación de solicitudes TCP con atacantes* 184

Figura 121 *Configuración y ejecución TCP mixta* 185

Figura 122 *Generación de solicitudes TCP mixta* 185

Figura 123 *Captura de respuestas de las solicitudes TCP mixta* 186

Figura 124 *Captura de paquetes SYN utilizados por los usuarios para iniciar conexiones TCP legítimas y paquetes SYN generados por el ataque* 186

Figura 125 *Configuración y ejecución UDP con llamada normal* 188

Figura 126 *Captura Wireshark UDP con llamada normal* 188

Figura 127 *Configuración y ejecución UDP con llamada bajo ataque* 189

Figura 128 *Captura Wireshark de la llamada bajo ataque* 191

Capítulo I: Antecedentes

1.1 Tema

Detección de ataques DDoS de baja intensidad en SDN mediante análisis de entropía de flujos y modelos Machine Learning interpretables

1.2 Problema

La evolución de los ataques de denegación de servicio distribuido (DDoS) ha derivado en nuevas variantes de baja intensidad (*low-rate*), que se caracterizan por utilizar patrones de tráfico persistentes y sutiles para evadir los mecanismos tradicionales de detección basados en el volumen. A diferencia de los ataques masivos, estos ataques de tipo silencioso degradan la disponibilidad de la red de manera progresiva, sin generar picos evidentes de congestión (Sudar & Deepalakshmi, 2022). Esta tendencia representa un reto crítico, especialmente en redes definidas por software (SDN), donde el controlador centralizado, encargado de gestionar los flujos, se convierte en un blanco altamente vulnerable ante este tipo de amenazas (Reddy & Meera, 2024)

Las soluciones tradicionales, como los sistemas de detección de intrusos basados en firmas o reglas estáticas, presentan limitaciones sustanciales en entornos SDN dinámicos, debido a su alta tasa de falsos positivos cuando enfrentan tráfico legítimo con comportamiento variable (Mohammad Aladaileh Mohammed Anbar, 2021). Por una parte, aunque se han aplicado modelos de aprendizaje automático para incrementar la capacidad de detección, la mayor parte de los modelos funcionan de forma tipo “caja negra” (como las redes neuronales profundas), lo que complica la adopción práctica en contextos donde la transparencia y la explicabilidad son requeridas para la toma de decisiones razonadas (Rudin, 2019). Este contexto resulta especialmente delicado en las infraestructuras críticas, en las que las decisiones del sistema deben ser auditarles. Diferentes investigaciones han dejado constancia que emplear métricas de entropía,

como, para poner un ejemplo, la entropía de Réyni, puede sumar un poco de sensibilidad a la hora de detectar pequeñas alteraciones en la distribución del tráfico, que es muy interesante de cara a detectar ataques DDoS de tipo low-rate (Fioravanti et al., 2023). En este sentido, la aplicación de métricas en real-time quizás no esté muy extendida en los entornos de la investigación de tráfico, que no son ajenos a la escasa utilización de modelos de clasificación, pero robustos y explicables. Por otra parte, también la existencia de datasets públicos y realistas escasea para poder entrenar modelos para tráfico SDN, lo que tiende a no ayudar a crear sistemas de manera rigurosa y reproducible (Puranik et al., 2023).

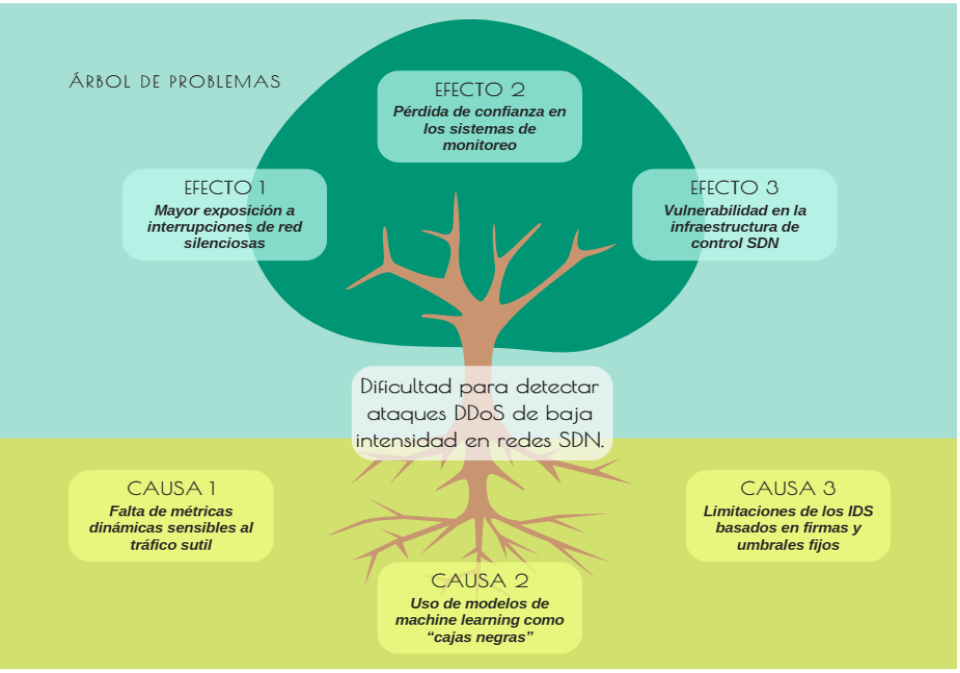
Ante esta problemática, se propone una aproximación híbrida basada en el análisis de entropía de flujos de red en tiempo real, combinada con el uso de modelos de machine learning interpretables, específicamente XGBoost junto con SHAP (Nadeem et al., 2023). Esta integración no solo busca detectar anomalías sutiles en flujos maliciosos de baja intensidad, sino también proporcionar explicaciones claras y comprensibles sobre las decisiones tomadas por el modelo. De este modo, se espera fortalecer la confianza en los sistemas automáticos de ciberseguridad y aportar una herramienta efectiva para redes SDN expuestas a amenazas persistentes.

En síntesis, la capacidad evasiva de los ataques DDoS de baja intensidad plantea un nuevo escenario de riesgo para las redes definidas por software, frente al cual los métodos tradicionales de detección resultan insuficientes. La falta de interpretabilidad de los modelos actuales y la carencia de métricas dinámicas aplicadas en tiempo real, constituyen brechas críticas que deben ser abordadas. En este contexto, la combinación de entropía de Rényi y machine learning explicable representa una alternativa viable y necesaria para avanzar hacia sistemas de detección más precisos, auditables y adaptables a las necesidades actuales de seguridad en entornos SDN.

Para mejor comprensión acerca del análisis que hacen referencia al planteamiento del problema se puede visualizar la Figura 1, en donde se detalla el árbol de problemas respectivo al tema de investigación.

Figura 1

Árbol de problemas ataques DDoS de Baja intensidad



1.3 Objetivos

1.3.1 *Objetivo General*

Implementar un sistema en redes SDN que permita la detección de ataques DDoS de tipo low-rate, mediante el análisis de entropía de flujos y modelos de Machine Learning interpretables.

1.3.2 *Objetivos Específicos*

- Analizar el estado del arte sobre técnicas de detección de ataques DDoS de tipo low-rate en redes SDN, con énfasis en el uso de entropía y modelos de machine learning interpretables.

- Diseñar un modelo de detección basado en análisis de entropía de flujos y aprendizaje automático explicable, adecuado para identificar tráfico anómalo de baja intensidad en entornos SDN.
- Desarrollar la propuesta en un entorno de simulación, integrando el cálculo dinámico de entropía y el modelo de clasificación entrenado.
- Evaluar la efectividad del sistema mediante pruebas con el modelo de detección entrenado y métricas de rendimiento, validando su capacidad de detección frente a ataques DDoS low-rate.

1.4 Alcance

El actual trabajo de grado parte de un enfoque de aplicación de la investigación en ciberseguridad, particularmente, en entornos definidos por software (SDN). Parte de una preocupación muy real, las redes modernas son altamente programables y resultan ser muy flexibles, pero también exponen a los sistemas a amenazas de sutilidad y detección difíciles. Una de las que persisten en el tiempo son los ataques DDoS de baja intensidad (low-rate) que, a pesar de no generar alarmas evidentes, comprometen la disponibilidad, integridad y, hasta, la confidencialidad de los sistemas. El objetivo principal es implementar un sistema de detección temprana que combine análisis estadístico y aprendizaje automático explicable, aportando claridad donde antes solo había incertidumbre. La solución propuesta será diseñada e implementada dentro de un entorno virtual controlado, incorporando el análisis estructurado de amenazas mediante la metodología STRIDE.

Para el enfoque metodológico se usará la metodología STRIDE como marco principal, ya que permite abordar de forma estructurada todas las fases relacionadas con la seguridad de sistemas. La propuesta considera seis etapas clave en el desarrollo del análisis y del diseño: la

identificación de lo que es el activo y el componente del sistema, la identificación de qué son las amenazas en función de las categorías STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service y Elevation of Privilege), el análisis de riesgos sobre lo que son los activos, el diseño de contramedidas, el desarrollo e implementación de un mecanismo de detección, y el monitoreo y mantenimiento del mecanismo que hemos implementado. Cada una de estas etapas desempeña un papel fundamental en la protección de la red SDN simulada bajo una arquitectura de centro de datos de tres niveles, al permitir identificar vulnerabilidades, evaluar riesgos y aplicar medidas de seguridad efectivas. Tomando como base esta metodología, se estructuran las fases del presente proyecto de investigación de la siguiente manera:

La primera fase del desarrollo del proyecto corresponde a la fundamentación teórica, orientada a la recopilación de información bibliográfica actualizada que permita contextualizar la problemática de seguridad en redes definidas por software (SDN), específicamente referidos a los ataques de denegación de servicio distribuidos de baja intensidad (DDoS low-rate); en dicho sentido, serán revisados diferentes enfoques de detección de ataques que se fundamenten en métricas estadísticas, poniendo el acento sobre todo en el uso de la entropía de Rényi, dada su capacidad para detectar pautas de tráfico que exhiben variaciones sutiles de (Yu et al., 2024).Paralelamente, se recuperarán las técnicas de inteligencia artificial enmarcadas en la interpretabilidad, centrándose en el uso de ciertos tipos de modelos, como XGBoost, y su correspondiente aplicación a SHAP,G los cuales permiten ofrecer resultados explicables y que se adecuan a entornos en los que se requiere que las decisiones sean comprensibles para los usuarios (Uysal & Kose, 2024)..Dicha revisión persigue poder dar forma a una base teórica que sustente la propuesta técnica del sistema de detección a implementar. Para llevarlo a cabo también se incluirá un análisis estructurado de amenazas sobre la arquitectura SDN presentada, usando la metodología

STRIDE, que permitirá poder identificar vulnerabilidades específicas, clasificar los vectores de ataque en función de su naturaleza y severidad y poder justificar la necesidad de incluir mecanismos preventivos adecuados que fortalezcan la seguridad del plano de control y, a su vez, mejoren la disponibilidad del sistema frente a ataques silenciosos y de difícil detección.

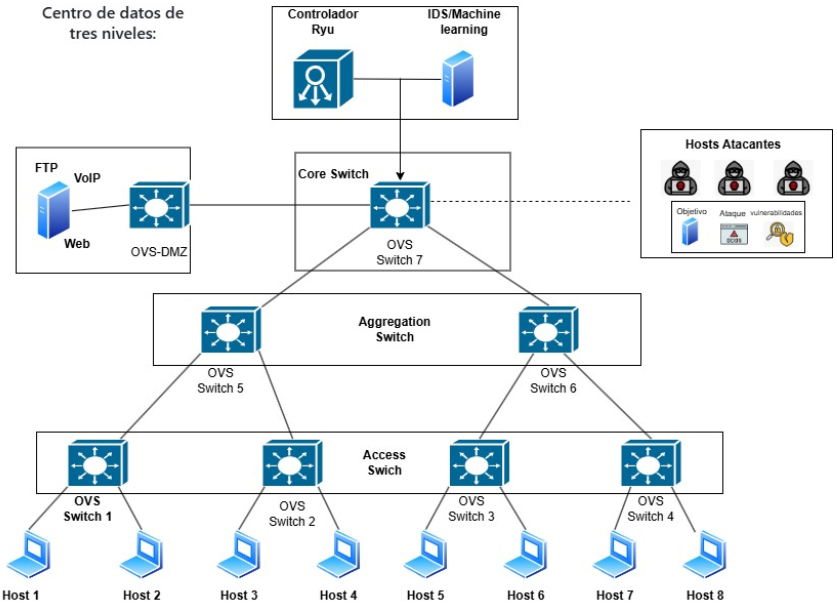
la segunda fase es el diseño e implementación de una arquitectura lógica de red SDN simulada que represente el modelo de una topología de centro de datos de tres niveles: acceso, agregación y núcleo. Para ello, se contará con una plataforma de simulación open-source que permita modelar el comportamiento de los componentes de red y, a su vez, facilite la integración de los módulos de análisis que se requieran para el sistema de detección. La arquitectura estará conformada por switches OpenFlow interconectados de manera jerárquica, por un controlador SDN Ryu que centraliza la gestión de la red y por un módulo externo de procesamiento que será desarrollado en Python y que será el encargado del cálculo dinámico de la entropía de Rényi y de aplicar el modelo de clasificación previamente entrenado con XGBoost y SHAP. La red contará con múltiples hosts legítimos en la capa de acceso, los cuales generarán tráfico normal hacia servicios simulados ubicados en niveles superiores, representando así un entorno realista de operación.

Una vez que la red se encuentre operativa, se establecerá el escenario de pruebas, el cual se ilustra en la Figura 2. Este entorno será utilizado para recrear las condiciones necesarias para evaluar el comportamiento del sistema frente a eventos normales y ataques maliciosos. En esta etapa, se deja preparada la estructura para que, en fases posteriores, un grupo de hosts simulados actúe como atacantes, con el fin de ejecutar tráfico tipo DDoS low-rate dirigido específicamente al núcleo de la red. Dicho tráfico permitirá verificar si el sistema de detección es capaz de

identificar las anomalías sutiles que caracterizan este tipo de amenazas, y cómo responde el modelo en tiempo real frente a dicha actividad anómala.

Figura 2

Arquitectura correspondiente al escenario de evaluación del sistema de detección.



Nota: Diseño basado en un centro de datos de tres niveles, utilizado para validar la detección de ataques DDoS low-rate en entornos SDN simulados.

La tercera fase del proyecto corresponde a la implementación técnica del escenario de pruebas definido previamente, utilizando como entorno principal la herramienta GNS3, ampliamente reconocida en la validación de arquitecturas experimentales en redes SDN (Lara et al., 2014a).Esta etapa se centra en desplegar la topología lógica diseñada, así como en integrar las herramientas necesarias para el funcionamiento del sistema de detección propuesto. A continuación se procederá con la configuración de la infraestructura de red que compone el centro de datos de tres niveles, y la puesta en marcha del controlador Ryu. Asimismo, se establecerá la comunicación con el módulo de análisis externo desarrollado en Python, donde se calculará la entropía de Rényi y se enviarán los valores extraídos al modelo de clasificación XGBoost que será

explicado mediante SHAP. Para capturar y analizar el tráfico de red se usarán Wireshark y CICFlowMeter para la extracción de atributos significativos del flujo. El sistema estará capacitado para enseñar a reconocer patrones en el tráfico ascendente proveniente de hosts legítimos, para procesarlos en contraposición a tráfico malicioso generado de manera manual por un conjunto de hosts atacantes que ejecutarán ataques DDoS de bajo volumen dirigidos, particularmente, al Core switch o a servicios críticos ubicados en la DMZ. Esta etapa se desarrolla en línea con las primeras etapas de la metodología STRIDE, donde se valida tanto los activos vulnerables como los vectores de ataque detectados y crear un entorno controlado en el que se mide el impacto de los ataques sobre la red. De esta manera, se sientan las bases para la fase siguiente, donde se evaluará el rendimiento del sistema de detección y su capacidad de operación en tiempo real, sin comprometer la disponibilidad ni la estabilidad del plano de control.

La cuarta y última fase del proyecto corresponde al proceso de validación del sistema de detección propuesto, ejecutado bajo condiciones activas dentro del entorno de simulación. En esta etapa, se pondrá a prueba el modelo previamente entrenado con un dataset personalizado, construido a partir de la adaptación de registros provenientes de fuentes públicas y ajustado para representar de forma realista características asociadas a ataques DDoS de baja intensidad. El propósito de esta fase es evaluar cómo se comportó el sistema ante eventos de tráfico anómalo que generaron los hosts maliciosos especificados dentro de la arquitectura y comprobar su capacidad de detectar ataques desde la primera parte del proceso, en tiempo real. Evaluaremos el sistema con un conjunto de métricas cuantitativas como son precisión, recall, puntuación F1 y tasa de falsos positivos para impulsar la cuantificación del rendimiento general del sistema. Por otro lado se aplicará una evaluación interpretativa con RHAP, que puede utilizarse para examinar el comportamiento del modelo e identificar qué atributos juegan un papel más importante en la toma

de decisiones del mismo. De forma más concreta, se verá cómo la entropía de Rényi se encuentra estructurada en su rol de variable explicativa dentro del proceso de clasificación, a través de la naturaleza de métrica de detección ante leves alteraciones en el flujo del tráfico. La fase de validación también incluye la reflexión crítica a partir de los resultados presentados, comparando el comportamiento de la red con el sistema de detección activo en relación con el que se comportaría en una red supuesta sin un sistema de protección, facilitando con esto la visualización de las mejorías conseguidas en términos de visibilidad, explicabilidad y robustez del plano de control que contiene a la red. Para finalizar se plantearán recomendaciones orientadas hacia un mantenimiento y monitoreo del sistema que perdura en el tiempo, incluyendo criterios para su integración en redes reales en su futuro y la actualización de sus modelos de detección ante nuevos tipos de amenazas emergentes.

1.5 Justificación

Esta propuesta de trabajo de titulación se considera perfectamente válida dada la situación de transformación digital por la que atraviesan las infraestructuras tecnológicas actuales; las redes definidas por software (SDNs) se han convertido en una eficiente opción para gestionar el tráfico de red de forma programable y centralizada. Sin embargo, esta misma flexibilidad viene acompañada de nuevos retos no solamente para la gestión del tráfico de red, sino también para la seguridad, especialmente en lo que respecta al plano de control, el cual puede acabar siendo un objetivo de alto impacto frente a problemas persistentes como son los ataques DDoS de baja intensidad (low-rate). Éstos, ya que los ataques DDoS de baja intensidad (low-rate) tienen un bajo número de flujos, aunque siempre constantes para conseguir degradar el rendimiento de la red sin necesidad de sobrepasar los umbrales convencionales de detección, representan una amenaza silenciosa pero crítica. (Agrawal & Tapaswi, 2021)

Por este motivo se hace necesario la generación de sistemas de detección temprana que combinen ambos métodos (métodos estadísticos y de aprendizaje automático para identificar en tiempo real patrones anómalos). En concreto, la entropía de Rényi ha resultado ser una métrica adecuada para detectar alteraciones en la distribución de tráfico como resultado de ataques low-rate (Puranik et al., 2023), y su combinación con modelos de machine learning interpretables, como XGBoost junto a SHAP, puede ayudar no solamente a realizar una clasificación adecuada sino a dar explicaciones de cada decisión del modelo, algo que resulta clave en entornos donde es fundamental dar un rastro y una transparencia. (Lundberg & Lee, 2017)

Este enfoque técnico se alinea directamente con el Objetivo de Desarrollo Sostenible n.º 9 (Industria, innovación e infraestructura), que promueve el fortalecimiento de sistemas resilientes, fiables y sostenibles, incluyendo la infraestructura digital crítica (Unidas, 2030) Asimismo, se enmarca dentro de los lineamientos de la Agenda de Transformación Digital del Ecuador, que establece como prioridad la incorporación de tecnologías emergentes orientadas a la detección y respuesta frente a amenazas cibernéticas. Con esto, el proyecto también contribuye a la mejora de la seguridad en las redes inteligentes y a la mejora de la capacidad técnica.

La solución presentada será validada a través de un entorno simulado controlado orientado a reproducir las condiciones operativas reales que puede tener una red SDN, generando tráfico legítimo y tráfico malicioso, todo ello con el fin de evaluar toda la capacidad de detección de la solución propuesta. Esta validación nos permitirá no solo contrastar su eficacia, sino también poder sentar las bases futuras para poder llevar la solución a un entorno productivo. Este trabajo no solo propone una solución innovadora frente a una amenaza técnica concreta, sino que plantea un modelo replicable, auditable y explicable para el desarrollo de sistemas de ciberseguridad basados en inteligencia artificial.

Capítulo II: Fundamentación Teórica

En este capítulo se abordan los fundamentos teóricos necesarios para el diseño del sistema de detección propuesto. Se analiza la ciberseguridad con énfasis en la disponibilidad como parte del modelo CIA y en los riesgos asociados a los ataques DDoS de baja intensidad en entornos SDN. Asimismo, se revisan las vulnerabilidades de estas redes, la necesidad de mecanismos de defensa inteligente, y el uso de la entropía de Rényi como métrica para identificar tráfico anómalo. Finalmente, se presentan los modelos de machine learning interpretables, junto con las métricas empleadas para evaluar el desempeño del sistema en entornos simulados.

2.1 Ciberseguridad

Se entiende por ciberseguridad como un campo multidisciplinario dedicado a prevenir la exposición de sistemas, redes y datos ante amenazas y vulnerabilidades que podrían comprometer el núcleo de la seguridad de la información. Esto se lo puede afirmar debido a que de acuerdo con Solms y Niekerk (2013) mencionan que la ciberseguridad ha pasado de ser una preocupación técnica única a un enfoque integrado con componentes tecnológicos, humanos, organizacionales e incluso estratégicos, lo que la convierte en un recurso indispensable para la sostenibilidad funcional y la continuidad operativa en la era impulsada digitalmente. Este aspecto se la debe principalmente debido a la creciente sofisticación de los ataques, las complejas arquitecturas de red y la creciente interconectividad global de los sistemas refuerzan e impulsan este tipo de enfoque holístico.

Para comprender la evolución en términos conceptuales que tiene la ciberseguridad se debe en primera forma entender que ha ido acompañada del establecimiento de estándares y normas de tipo internacional como se lo puede evidenciar en la familia ISO/IEC 27000, que se encuentra enfocado en la implementación de un Sistema de Gestión de Seguridad de la Información (SGSI),

con la finalidad de estar centrado en la evaluación y tratamiento del riesgo. Por otro lado, también se puede mencionar el NIST Cybersecurity Framework (CSF), el cual fue desarrollado por el Instituto Nacional de Estándares y Tecnología de los Estados Unidos para plantear cinco funciones clave para la gestión del ciclo de vida de la seguridad, que suelen ser entre identificar, proteger, detectar, responder y recuperar (Institute of Standards, 2024)

2.1.1 Seguridad de la información

La seguridad de la información corresponde a la práctica de resguardar los activos de información ante amenazas que comprometan su confidencialidad, su integridad y su disponibilidad (CIA), a través del acceso a políticas, controles técnicos y procesos organizativos. Es decir, que estos mecanismos garantizan que solo usuarios autorizados accedan a los datos con la particularidad de que la información permanezca precisa, completa y sobre todo disponible en cualquier momento que se lo requiera, alineándose con las mejores prácticas del campo (Von Solms & Van Niekerk, 2013).

Como se pudo evidenciar para que exista una perfecta protección de la información se hace uso de aspectos regulatorios, entre los cuales se puede destacar el estándar ISO/IEC 27001:202 establecido como referente principal internacional en cuanto a SGSI que hace uso del ciclo de mejora constante, la cual se encuentra comprendida entre planificar, hacer, verificar y actuar (PHVA). Este aspecto se la describe de tal manera debido a que la literatura respalda su adopción por parte de empresas de múltiples sectores que buscan fortalecer su postura frente a amenazas digitales (Institute of Standards, 2024).

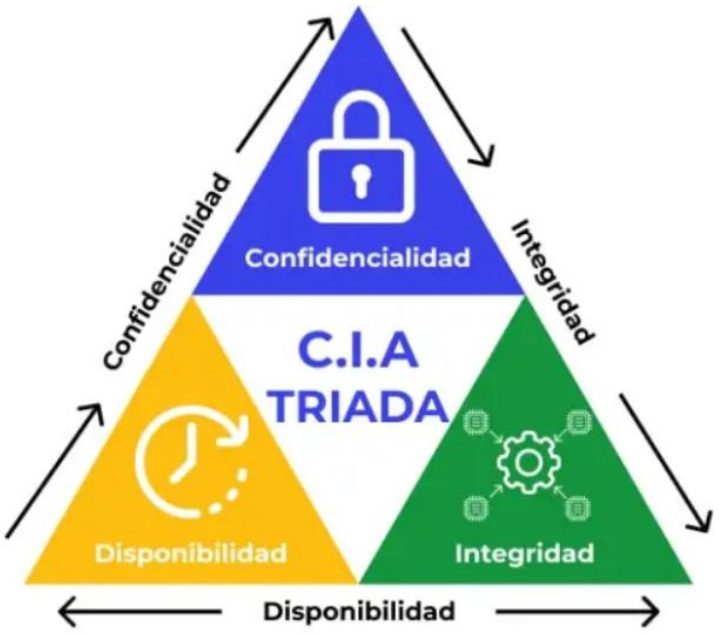
2.1.2 Modelo CIA

Este tipo de modelo se la puede interpretar como el pilar principal en cuanto a la seguridad de la información, debido a que permite definir tres aspectos importantes que no deben faltar en

cualquier tipo de sistema. Dentro de esto se puede mencionar la confidencialidad para garantizar que los usuarios específicamente autorizados puedan acceder a la información, la integridad para asegurar que los datos no puedan ser alterados sin autorización y por último la disponibilidad para que la información sea accesible en cualquier momento deseado (Chai & Zolkipli, 2021). Esto se lo puede evidenciar claramente en la siguiente Figura 3.

Figura 3

Triangulo C.I.A



Nota: Obtenido de (Wallarm, n.d.)

Como se pudo observar en la Figura 3 los tres principios de confidencialidad, integridad y disponibilidad se encuentran relacionados entre sí para asegurar la protección de cualquier tipo de activo informático. Al ser principios fundamentales al presentar una violación de uno de estos puede comprometer la seguridad completa del sistema, por lo que su gestión debe ser prioritaria y cuidadosa en cada aspecto que se encuentre sin importar el entorno en el que se encuentre, ya sea empresarial, gubernamental o académico.

2.1.3 Ataques

Los ataques a los sistemas de información son actos maliciosos que se llevan a cabo deliberadamente para comprometer activos digitales que afectan la confidencialidad, integridad o disponibilidad de la información. Tales amenazas no solamente se limitan a phishing y malware para afectar la vulnerabilidad sino también pueden presentarse como ataques complejos como las amenazas persistentes avanzadas (APT) para lograr su objetivo de afectar el modelo CIA de la información. (Agrafiotis et al., 2018)

Al llamarse ataque por lo general genera daños y esto se lo puede evidenciar de mejor manera a través de la investigación realizada por (Agrafiotis et al., 2018) se pudo describir que al presentar ciberataques hacia la disponibilidad como los DDoS y la integridad por medio de malware compromete enormemente la infraestructura volviéndola crítica y peligrosa. (Agrafiotis et al., 2018). En la siguiente Tabla 1 se pueden observar los diferentes tipos de ataques que existen en las redes.

Tabla 1

Tipos de Ataques

Tipo	Descripción	Ejemplos	Capa principal	Detección / mitigación
DoS	Un único origen agota recursos/red y vuelve el servicio indisponible.	Floods simples (ICMP/UDP), Slowloris de un host.	Red	Límites de conexiones, timeouts.
DDoS	Múltiples orígenes coordinados tráfico masivo que complican el bloqueo.	multi-origen.	Multicapa	Plan de respuesta, telemetría, coordinación con ISP.

Tipo	Descripción	Ejemplos	Capa principal	Detección / mitigación
Volumétricos	Buscan saturar ancho de banda con gran caudal (bps).	UDP/ICMP floods, amplification a gran escala.	Red	Filtrado en borde.
Low-rate	Tráfico de baja tasa que puede colgar servicios.	Slow HTTP, conexiones incompletas sostenidas.	App / sesión	Detección por patrones temporales, timeouts agresivos.

Nota: Adaptado de (Khan et al 2022) y (Stallings, 2020).

2.1.3.1 Ataques DoS. Este aspecto hace referencia a que los ataques de denegación de servicio (DoS) se enfocan en el envío masivo de solicitudes hacia un solo objetivo para agotar los recursos del sistema y generar fallas en cuanto a disponibilidad de forma temporal (Upadhyay & Rathee, 2022). Este tipo de ataque a pesar de ser menos complejo que los ataques distribuidos, todavía causan un inmenso impacto en la economía, ya que puede generar que se pierda dinero por cada minuto de parada en sistemas de sectores como el comercio electrónico y servicios financieros**.**

Esto se lo puede afirmar gracias a la investigación de (Joosten & Nieuwenhuis, 2017) , donde se señala que los ataques DoS en entornos bursátiles, al provocar interrupciones en los servicios, pueden generar caídas significativas en el valor de las acciones de la empresa afectada, especialmente cuando dichas interrupciones impactan directamente en la experiencia del cliente. Como se observó en la Tabla 1 se detallan los diferentes tipos de ataques DoS, lo que permite comprender su diversidad y el potencial impacto que pueden tener sobre la infraestructura tecnológica.

2.1.3.2 Ataques DDoS. Los ataques de denegación de servicio distribuido (DDoS) impactan a las víctimas a un nivel exponencial al abrumarlas desde múltiples fuentes. Ejemplos claros de esta afectación se las puede evidenciar en el informe de (Gcore, 2024) en donde describe que los ataques DDoS han ido en aumento en cuanto a daños como el compromiso de la información en base a picos de trafico de hasta 1.7 Tbps. En cuanto a pérdidas económicas según (Corero, 2024) mediante un reporte evidencia que al presentar un ataque de este tipo genera daños económicos de alrededor de $6 000 por minuto mientras se encuentra interrumpiendo las actividades.

2.2 Redes Definidas por Software (SDN)

Las Redes Definidas por Software SDN por sus siglas en inglés constituyen una arquitectura avanzada de gestión de redes que introduce un modelo de operación basado en la separación lógica entre el plano de control y el plano de datos. Esta división permite que las decisiones de control se centralicen en un componente denominado controlador, mientras que los dispositivos de red, como los switches, se encargan únicamente del reenvío de paquetes. De acuerdo con (Kreutz et al., 2015) este enfoque puede generar varios factores positivos como establecer una red más flexible, escalable y programable al permitir que las políticas de control puedan modificarse de forma dinámica mediante el uso de software sin tener la necesidad de alterar la infraestructura física subyacente.

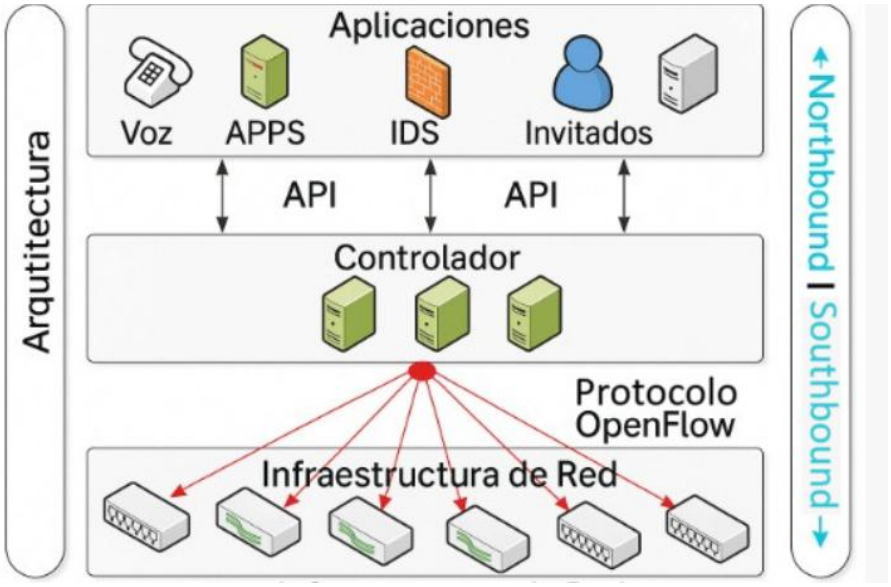
2.2.1 Arquitectura SDN

La arquitectura SDN se la puede encontrar en una estructura compuesta por tres capas, las cuales van desde la aplicación, control e infraestructura. Esta forma de estructuración se fundamenta en la separación lógica entre el plano de control y el plano de datos, con la finalidad de permitir la toma de decisiones hacia un componente de software, mientras que los dispositivos

de red se encargan específicamente del reenvío de paquetes. De acuerdo con (Lara et al., 2014b) este modelo facilita una administración más eficiente al simplificar la implementación de políticas, automatizar procesos y permitir una adaptación dinámica frente a cambios operativos o amenazas emergentes. En la siguiente Figura 4 se puede detallar la arquitectura base que suelen llegar a contemplar las redes SDN.

Figura 4

Arquitectura de red basada en SDN



Nota: Obtenido de (Tiweari,V, 2013)

Como se observó en la Figura 4 la arquitectura de SDN se encuentra compuesta por tres capas funcionales, de las cuales van desde la capa de aplicación, la capa de control y la capa de infraestructura interactuando entre sí a través de interfaces estandarizadas para permitir un diseño modular y flexible. En lo que respecta la funcionalidad de cada capa se puede mencionar lo siguiente, la capa de aplicación se encuentra ubicada en la parte superior de la arquitectura conformada por un conjunto de aplicaciones que expresan políticas, objetivos y requerimientos funcionales de la red, con la presencia de componentes que incluyen herramientas de monitoreo,

orquestación, balanceo de carga, control de acceso y gestión de seguridad. Estas aplicaciones interactúan con el plano de control a través de la interfaz norte (Northbound API) la cual proporciona un canal lógico que abstrae la complejidad de la infraestructura subyacente. Según el RFC 7426 esta interfaz permite que los desarrolladores de aplicaciones definan comportamientos de red sin depender del hardware, promoviendo un diseño modular, interoperable y altamente adaptable a los requisitos del entorno operativo.

En lo que respecta a la capa de control se puede mencionar que corresponde al núcleo lógico del sistema SDN, ya que se encuentra compuesta por uno o varios controladores encargados de traducir las políticas definidas en la capa de aplicación en instrucciones específicas que serán ejecutadas por los dispositivos de red. Este factor mantiene una vista global del estado de la red basada en la recopilación continua de datos provenientes del plano de infraestructura, en donde la comunicación descendente se la realiza mediante la interfaz sur (Southbound API) utilizando protocolos como OpenFlow, Netconf o P4.

Este aspecto de la capa de control se lo puede validar de acuerdo con (Benzekki et al., 2017) ya que mencionan que este tipo de capa introduce un modelo centralizado de toma de decisiones que mejora la eficiencia en la gestión de recursos, reduce la complejidad operativa y permite una respuesta rápida ante fallos o ataques.

Por último, se encuentra la capa de infraestructura considerada el plano de datos de la arquitectura SDN, ya que se encuentra conformada por los dispositivos físicos o virtuales que ejecutan funciones de reenvío de paquetes. Estos dispositivos como switches, routers o puntos de acceso no toman decisiones por sí mismos, sino que actúan conforme a las reglas impuestas desde el plano de control. De acuerdo con el informe técnico de la RFC 7426 descrito por (Haleplidis et al., 2015) describe que esta capa se comunica con el controlador a través de canales seguros y

permite la reconfiguración dinámica de la red sin interrumpir el servicio, lo cual es esencial para garantizar alta disponibilidad y eficiencia en redes modernas.

2.2.2 Protocolo OpenFlow

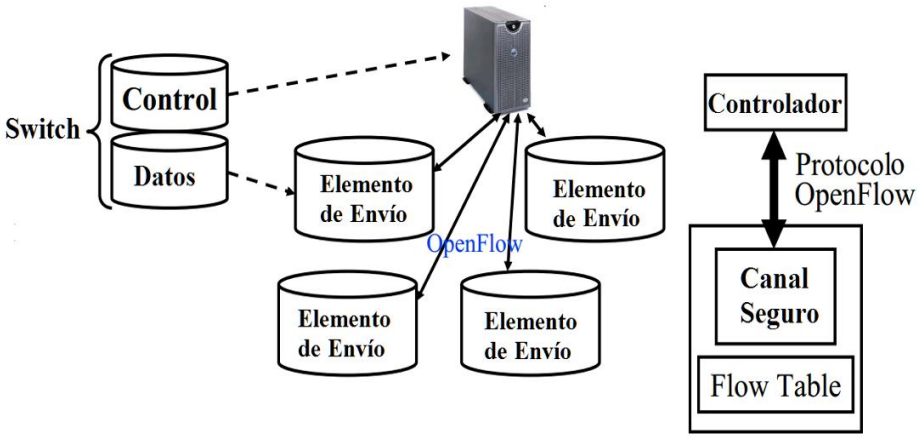
El protocolo OpenFlow se la puede considerar en uno de los pilares fundamentales de la arquitectura SDN, debido a que permite la comunicación directa entre el controlador central y los dispositivos de red independientemente de que sean físicos o virtuales. Su principal función es habilitar la programación dinámica del plano de datos a través de una interfaz estándar y abierta. OpenFlow fue propuesto inicialmente por la Universidad de Stanford y posteriormente estandarizado por la Open Networking Foundation (Open Networking Foundation, 2013), con la finalidad de conformarla a través de 3 elementos principales.

En primer lugar, se encuentra el controlador SDN considerado como un software centralizado encargado de establecer y mantener la comunicación con los dispositivos de red mediante canales seguros. El controlador define políticas de red y gestiona el reenvío, configurando dispositivos según el estado de la red. Se observan las tablas de flujo en los switches, que contienen las reglas para el manejo de paquetes. Las reglas incluyen campos de coincidencia, contadores y acciones como reenviar, descartar o modificar paquetes.

Por último, se puede visualizar los conmutadores SDN considerados los dispositivos tanto físicos como virtuales que se encargan de ejecutar las acciones definidas por el controlador. Estos dispositivos pueden operar únicamente bajo instrucciones, ya que no constan con la capacidad de tomar decisiones de forma automatizada. Para mayor visualización se puede referenciar a la Figura 5 para comprender como se encuentran subdivididos dicha arquitectura de red.

Figura 5

Arquitectura de red basada en SDN



Nota: Obtenido de (Índice 1. OpenFlow y sus herramientas 2. Software Defined Networking (SDN) 3. Network Function Virtualization (NFV), s/f)

Como se pudo observar en la Figura 5 puede ir desde la separación entre el plano de control y el plano de datos, así como el uso de un canal seguro para el intercambio de mensajes entre el controlador y los dispositivos de red mediante el protocolo OpenFlow. Una característica que se puede notar es que este tipo de canal permite que el controlador modifique dinámicamente las tablas de flujo en función del tráfico observado, habilitando así un comportamiento de red más flexible, centralizado y programable.

2.2.2.1 Mensajes OpenFlow. Este tipo de aspecto hace referencia que la comunicación entre el controlador y los dispositivos de red se las realiza mediante un conjunto estandarizado de mensajes que permiten la gestión del tráfico, la instalación de reglas, la recopilación de estadísticas y la detección de errores. Por lo que se considera que estos mensajes son esenciales para mantener una sincronización adecuada entre el plano de control y el plano de datos, así como también lograr permitir una gestión reactiva y dinámica de la red. Para mejor visualización de los diferentes mensajes que se presentan la siguiente Tabla 2 detalla de mejor manera los diferentes tipos que se puede llegar a encontrar**.**

Tabla 2

Tipos de mensajes OpenFlow

Tipo de mensaje	**Dirección**	**Descripción**
Packet-In	Switch → al Controlador	Se genera cuando un paquete no coincide con ninguna regla en la tabla de flujo.
Packet-Out	Controlador → al Switch	El controlador enseña al switch para reenviar un paquete específico a un puerto determinado.
Flow-Mod	Controlador → al Switch	Permite añadir, modificar o eliminar reglas en la tabla de flujo del switch.
Stats Request	Bidireccional	Solicita y recibe estadísticas sobre el tráfico.
Hello	Bidireccional	Establece la conexión inicial entre el controlador y el switch.
Features Request	Bidireccional	El controlador consulta las capacidades del switch para adaptarse a su estructura.
Error	Switch → al Controlador	Notifica fallos en el procesamiento de mensajes OpenFlow.

Nota: Adaptado de (Open Networking Foundation, 2015) y (Lara et al, 2014).

Como se observó en la Tabla 2 cada uno de los mensajes cumple una función en específico durante la operación de la red. Dentro de estos mensajes se puede considerar dos tipos que se los puede considerar sumamente relevantes, los cuales son los Packet-In y Flow-Mod considerados de

esta forma debido a que forman parte fundamentalmente en la implementación de sistemas de detección basados en el comportamiento del tráfico, además de permitir la inspección en tiempo real y la aplicación de reglas personalizadas en respuesta a eventos específicos.

2.3 Seguridad en SDN

La arquitectura de las SDN representa un cambio significativo respecto a los enfoques tradicionales, debido a que ofrecen un plano de control centralizado y una gestión más dinámica del tráfico. Llegando a establecerse una flexibilidad, pero también introduce nuevos desafíos de seguridad, debido a la separación entre el plano de control y el de datos, así como la dependencia del canal de comunicación entre el controlador y los dispositivos de red, permitiendo que se amplie la superficie de ataque (Scott-Hayward et al., 2016).

2.3.1 Vulnerabilidades en SDN

El diseño modular de SDN se la puede considerar beneficioso para la administración y orquestación de redes, pero puede llegar a exponer múltiples vectores de ataque, los cuales pueden ser aprovechados por personas maliciosas generando varias amenazas, pero las más comunes son las que se describen a continuación.

2.3.1.1 Desbordamiento de la tabla de flujo. Los switches OpenFlow utilizan memorias TCAM (Memoria Ternaria Direccionable por Contenido) para almacenar reglas de flujo, consideradas costosas y limitadas en capacidad, por lo que un atacante puede aprovechar esta falencia logrando enviar grandes cantidades de flujos falsos. Dentro de este aspecto se denota que cada nuevo flujo provoca que el controlador instale una nueva regla, lo cual rápidamente satura la tabla y una vez llena el switch no puede atender nuevos flujos a pesar de que provengan de usuarios legítimos, generando así una denegación de servicio (Scott-Hayward et al., 2013)**.**

2.3.1.2 Congestión del enlace de control y datos

Cuando un paquete no coincide con ninguna regla en la tabla de flujo del switch, este genera un mensaje Packet_In que es enviado al controlador para recibir instrucciones. De acuerdo con esta acción los atacantes pueden abusar de esta mecánica inyectando paquetes con direcciones IP falsas para provocar una avalancha de Packet_In, logrando así saturar el canal de comunicación entre el plano de datos y el plano de control. Esta forma de ataque reduce la capacidad de respuesta del controlador y degrada el rendimiento de la red (Shin & Gu, 2013).

2.3.1.3 Saturación del controlador. El controlador SDN es un componente centralizado que debe tomar decisiones en tiempo real sobre todos los flujos que no tienen reglas predefinidas. Un volumen elevado de flujos falsos o manipulados puede sobrecargar la capacidad computacional del controlador, afectando su capacidad para procesar solicitudes legítimas. Esto resulta en pérdida de paquetes, latencia elevada e interrupciones en el servicio (Kreutz et al., 2019).

2.3.2 Ataques DDoS de baja intensidad en SDN

Los ataques DDoS de baja intensidad se las puede considerar también como ataques de tipo low-rate o tipo slow, establecidas como amenazas sofisticadas y de difícil detección en cuanto a entornos SDN, debido a que utilizan flujos aparentemente legítimos, pero diseñados para

degradar la calidad del servicio sin exceder umbrales típicos de detección (Pham et al., 2021). A diferencia de los ataques DDoS tradicionales estos buscan saturar la red con tráfico volumétrico explotando las características propias del plano de control como el procesamiento de mensajes Packet-In, para generar una carga excesiva en el controlador sin levantar sospechas inmediatas.

Estudios recientes han señalado que los ataques DDoS de baja intensidad en SDN no solo afectan la disponibilidad, sino que también generan inconsistencias en la sincronización de los controladores distribuidos, ya que de acuerdo con (Ali et al., 2022) exponen la red a problemas de control lógico y fallos en la toma de decisiones globales. Por lo que ante esta amenaza se ha propuesto el uso de métricas de entropía, algoritmos de aprendizaje automático y enfoques de mitigación con SDN federadas para identificar patrones de tráfico anómalos sin comprometer la eficiencia operativa.

2.3.3 Sistemas IDS en SDN

La implementación de sistemas de detección de intrusos (IDS) en entornos SDN ha permitido establecer relevancia en los últimos años como respuesta a las nuevas amenazas que surgen de la arquitectura centralizada de estas redes. Según Scott-Hayward et al. (2016) la separación entre el plano de control y el plano de datos, característica inherente a SDN, ofrece una oportunidad estratégica para integrar mecanismos de detección que operen desde el controlador, con una visión global del estado de la red.

Los sistemas IDS en SDN permiten una supervisión más eficiente del tráfico mediante la utilización de reglas dinámicas que pueden ser actualizadas en tiempo real. Lo cual se lo puede afirmar de acuerdo con Nanda et al. (2022) ya que sostienen que esta capacidad de respuesta adaptativa se ve potenciada al incorporar técnicas de aprendizaje automático, facilitando la detección de comportamientos anómalos con mayor precisión en comparación con los enfoques

tradicionales. Sin embargo, también advierten que pueden seguir persistiendo desafíos importantes relacionados con la tasa de falsos positivos, la sobrecarga del controlador y la necesidad de mantener la escalabilidad del sistema ante entornos de tráfico que pueden ir creciendo de forma compleja.

2.4 Modelos ML

Los modelos ML son esenciales en el campo de la inteligencia artificial y la ciencia de datos, debido a que permiten que las máquinas aprendan patrones a partir de datos sin la necesidad de ser programadas explícitamente. Pero para lograr esta funcionalidad necesitan utilizar algoritmos para identificar los patrones en grandes volúmenes de datos y pueden hacer predicciones o tomar decisiones basadas en esta información, lo que los hace aplicables al momento de desear detectar anomalías, clasificar datos y la regresión (Zhang & Li, 2020).

Estos tipos de modelos pueden llegar a ser divididos en dos categorías principales como los supervisados y no supervisados como se nos muestra en la siguiente Tabla 3. Los modelos supervisados hacen referencia a que requieren datos etiquetados para aprender, es decir, que el conjunto de entrenamiento incluye ejemplos con sus respectivas salidas correctas. En comparación con los no supervisados que no requieren datos etiquetados y buscan identificar estructuras o patrones inherentes en los datos como agrupamientos o relaciones ocultas (Alpaydin, 2020).

Tabla 3

Modelos Machine Learning

Modelo de ML	Tipo de aprendizaje	Aplicación en ciberseguridad	Ventajas destacadas	Referencias
XGBoost	Supervisado	Clasificación avanzada de tráfico, detección de ataques con alta precisión	Alta precisión, escalabilidad, manejo de datos faltantes y regularización	Chen, T., & Guestrin, C. (2016)
Random Forest (RF)	Supervisado	Clasificación de tráfico de red, detección de intrusiones	Alta precisión, robustez ante overfitting	Breiman, L. (2001)
Decision Tree (DT)	Supervisado	Interpretabilidad en clasificación de tráfico	Fácil interpretación, bajo costo de entrenamiento	Jijo, B. T., & Abdulazeez, A. M. (2021)
Support Vector Machine (SVM)	Supervisado	Separación de clases con márgenes óptimos	Buen rendimiento con alta dimensionalidad	Hao, P.-Y., Chiang, J.-H., & Chen, Y.-D. (2022)
K-Means	No supervisado	Detección de patrones anómalos, agrupamiento de tráfico	Simple, eficiente, útil para exploración inicial de datos	Ikotun, A. M., Abuhaija, B., & Heming, J. (2023)

Nota: Modelos Machine Learning comúnmente utilizados para ciberseguridad.

2.4.1 Modelos Machine learning Explicables

El desarrollo de modelos de Machine Learning ha consolidado su uso como una herramienta fundamental en la detección automatizada de amenazas informáticas, gracias a su capacidad para analizar grandes volúmenes de datos y detectar patrones anómalos. Sin embargo, el uso de algoritmos cada vez más complejos, como los modelos de ensamble o redes neuronales ha generado un problema para la toma de decisiones.

Es por eso la necesidad de interpretar los resultados generados y adentrar en el campo conocido como Explainable Machine Learning (XML). Esta área se enfoca en hacer comprensibles

las decisiones de los algoritmos, especialmente en escenarios donde la transparencia y la confianza en el sistema son fundamentales. En la Tabla 4 se presentan los dos enfoques principales de explicabilidad: los modelos interpretables, cuya estructura permite comprender directamente el proceso de decisión; y las técnicas de explicabilidad post hoc, que se aplican una vez entrenado el modelo para ofrecer interpretaciones sobre su comportamiento (Lundberg & Lee, 2017).

Tabla 4

Modelos de aprendizaje automático explicables

Enfoque de explicabilidad	Técnica / Modelo	Características principales	Referencias
Modelo interpretable	Árboles de decisión	Estructura jerárquica fácil de interpretar; reglas explícitas basadas en atributos.	Jijo, B. T., & Abdulazeez, A. M. (2021)
Modelo interpretable	Regresión logística	Modelo lineal con coeficientes que indican la influencia directa de cada variable.	Sperandei, S., Bastos, L. Reis, A., & Bastos, F. I. (2018)
Post hoc	SHAP	Basado en valores de Shapley; proporciona explicaciones locales y globales consistentes.	Lundberg, S. M., & Lee, S.-I. (2017)
Post hoc	LIME	Genera modelos simples locales para interpretar decisiones complejas.	Ribeiro, M. T., Singh, S., & Guestrin, C. (2016)

Nota: Adaptado de (Lundberg & Lee, 2017) y (Ribeiro et al, 2016)**.**

2.5 Teoría de la Entropía

Se la puede considerar útil al momento de realizar un análisis de redes SDN, con la finalidad de evaluar la incertidumbre y la distribución de probabilidades en el tráfico de red. Según Shannon (1948) la entropía puede llegar a medir la cantidad de información necesaria para describir un sistema o proceso aleatorio, la cual se la puede emplear tanto en el análisis del tráfico como en la detección de anomalías dentro de la red, permitiendo identificar patrones inesperados que podrían inducir a ataques.

2.5.1 Entropía de Shannon

La entropía de Shannon fundamentalmente mide la incertidumbre de un conjunto de eventos posibles en función de sus probabilidades. Es decir, dada una variable aleatoria X que puede tomar N valores posibles con la presencia de varias probabilidades, de tal manera la entropía se la puede definir de la siguiente manera mediante la siguiente Ecuación 1.

Ecuación 1

Formula de entropía de Shannon

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Donde:

- H(X): entropía de Shannon
- $P_{i:}$ probabilidad de que ocurra el evento x_i.
- Log_2: utilizado para medir la cantidad de información en bits.

En otras palabras, se puede llegar a interpretar que la Entropía es la cantidad promedio de información que se obtiene al observar un evento (X), cuando más impredecible o disperso es el sistema mayor será su entropía.

2.5.2 Entropía de Rényi

La entropía de Rényi propuesta por Alfréd Rényi en 1961 básicamente extiende la entropía de Shannon mediante la introducción de un parámetro (α), el cual ajusta la sensibilidad de la entropía a los eventos más probables, con la finalidad de ofrecer mayor flexibilidad en su aplicación como lo podemos observar en la siguiente Ecuación 2.

Ecuación 2

Formula de entropía de rényi

$$H_\alpha(X) = \frac{1}{1-\alpha}\log\left(\sum_{i=1}^{n} P_i^\alpha\right)$$

Donde,

- Hα(X): entropía de Rényi de orden α.
- P_i: probabilidad del evento X_i.
- α: ajuste a la sensibilidad de la medida a los eventos más probables.

Dentro de esto cuando α=1 se puede interpretar que entropía de Rényi se reduce a la entropía de Shannon, debido a que el parámetro (α) permite que la entropía de Rényi sea más sensible a ciertos tipos de eventos en comparación con la entropía de Shannon, lo que la hace útil en contextos donde se desea enfocar más en eventos raros o en la distribución de flujos de tráfico.

2.5.3 Entropía aplicada a redes SDN

En la siguiente Tabla 5 se presentan diversos estudios recientes que han aplicado técnicas de análisis de entropía en entornos de redes definidas por software (SDN). Estos trabajos evidencian cómo diferentes tipos de entropía, como la de Shannon o Rényi, han sido utilizadas para detectar anomalías en el tráfico de red, con enfoques que combinan estadística e inteligencia artificial.

Tabla 5

Estudios realizados con técnicas de entropía en SDN y Métricas Obtenidas.

Autor(es) y año	Título del estudio	Entropía	Aplicación	Precisión / Accuracy	Métricas reportadas	Herramientas
Ahalawat et al. (2022)	A low-rate DDoS detection and mitigation for SDN using Rényi Entropy with Packet Drop	Rényi	Detección y mitigación de DDoS de baja tasa en SDN mediante umbral de entropía y packet drop	98.3%	Accuracy, Precision, Recall, FPR	Mininet + Ryu; ataques simulados (hping3)
Fioravanti et al. (2023)	Entropy-based DDoS Detection in Software Defined Networks	Shannon	Detección de DDoS a partir de caídas abruptas de entropía en el plano de control SDN	No reporta precisión cuantitativa	Caídas de entropía, patrón de tráfico (gráficos)	Tráfico sintético en POX Controller
Kishansmaran et al. (2023)	A Two-level DDoS Attack Detection using Entropy and Machine Learning in SDN	Shannon	Detección jerárquica en dos niveles con análisis de entropía y Random Forest	Random Forest: 98.3%	Accuracy, F1-score, Confusion Matrix	Mininet (2000 benignos y 2000 ataques)

Nota: Elaboración propia con base en los estudios de ahalawat et al. (2022), Fioravanti et al. (2023) y kishansmaran et al. (2023).

2.5.4 Métricas de evaluación

Aquí se presentan las principales métricas de evaluación utilizadas para medir el rendimiento de los modelos de Machine Learning (ML). Estas métricas son esenciales para evaluar la precisión, eficacia y fiabilidad de los modelos en diversas aplicaciones, permitiendo una comprensión más profunda del comportamiento del modelo en función de los datos analizados. Las cuales necesitan inicialmente la consideración de los siguientes elementos de matriz de confusión para definir sus fórmulas matemáticas de forma exacta:

- **TP (True Positives)** representan los verdaderos positivos.
- **TN (True Negatives)** los verdaderos negativos.
- **FP (False Positives)** los falsos positivos.
- **FN (False Negatives)** los falsos negativos.

A partir de estos elementos las métricas respectivas para calcular de acuerdo con (Alashhab et al., 2023) son las siguientes:

2.5.4.1 Precisión. La precisión se define como una métrica que mide la exactitud de las predicciones positivas mediante la siguiente Ecuación 3**.**

Ecuación 3

Métrica de precisión

$$Precisión = \frac{TP}{TP + FP}$$

2.5.4.2 Recall (Sensibilidad). El recall es una métrica que nos indica cuantos casos positivos reales fueron correctamente detectados mediante la siguiente Ecuación 4.

Ecuación 4

Métrica de Sensibilidad

$$Recall = \frac{TP}{TP + FN}$$

2.5.4.3 F1-Score. Es una métrica balanceada que considera tanto falsos positivos como falsos negativos mediante la siguiente Ecuación 5**.**

Ecuación 5

Métrica de F1-Score

$$F - measure = 2 \cdot \frac{Precisión\ x\ Recall}{Precisión + Recall}$$

2.5.4.4 Exactitud. Esta métrica nos ayuda a medir la precisión global del modelo mediante la siguiente Ecuación 6**.**

Ecuación 6

Métrica de Exactitud

$$Exactitud = \frac{TP + TN}{TP + TN + FP + FN}$$

El cálculo de estas métricas se basó en los valores de la matriz de confusión previamente mencionados, garantizando así una evaluación cuantitativa del rendimiento para la sustentación de la eficacia del enfoque propuesto en la detección de ataques dentro de la arquitectura SDN.

Capítulo III: Despliegue y Funcionamiento

En el presente capitulo se encuentra detallado la metodología utilizada para el diseño de un modelo de detección basado en el análisis de entropía de flujos y aprendizaje automático explicable, adecuado para identificar tráfico anómalo de baja intensidad en entornos SDN. Todo esto mediante el establecimiento de requerimientos necesarios que cumplan con el sistema deseado mediante la norma ISO/IEE 29148 y para validar su calidad a través de la guía de la ISO/IEC 25010 en cuanto la compatibilidad, fiabilidad, seguridad, mantenibilidad, eficiencia de desempeño y usabilidad dentro de los estándares necesarios que se deben cumplir.

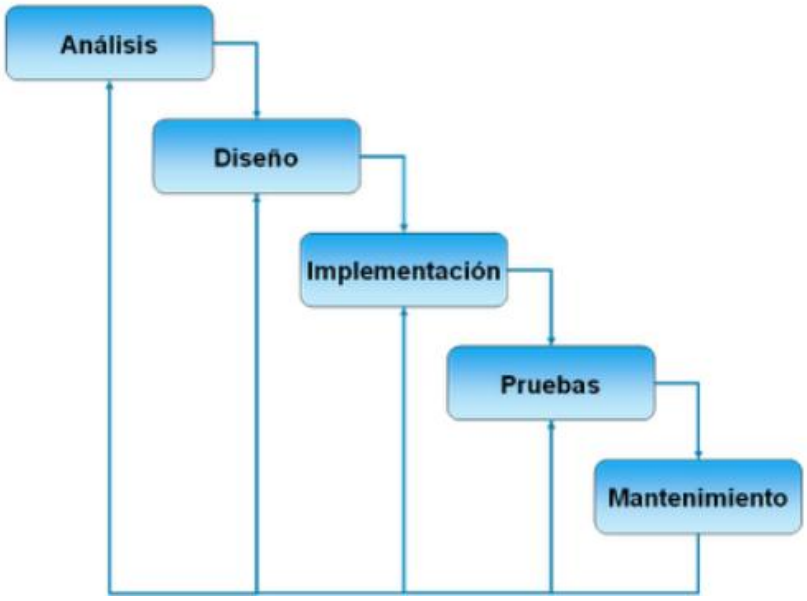
Además de presentar el desarrollo de su funcionamiento en diferentes escenarios con su respectiva evaluación acorde a la metodología de amenazas STRIDE enfocado especialmente en Denial of service (denegación de servicio).

3.1 Metodología

El modelo en cascada es una metodología secuencial que organiza el desarrollo en fases consecutivas como el establecimiento de requerimientos, diseño, implementación, pruebas y mantenimiento. Lo que lo vuelve idóneo para el propósito del trabajo que consiste en desarrollar un sistema de detección de ataques DDoS de baja intensidad mediante la entropía de flujos, debido a su carácter lineal que garantiza trazabilidad y control en cada etapa del diseño de forma técnica necesaria para el sistema deseado como se lo puede observar en la siguiente Figura 6.

Figura 6

Modelo en Cascada



Nota: Autoría Propia

3.2 Análisis

El análisis del modelo en cascada en esta investigación se orienta a identificar los requerimientos del sistema de detección de ataques DDoS de baja intensidad en SDN, considerando la necesidad de un entorno de simulación de redes, un controlador SDN, herramientas de captura y monitoreo de tráfico además de librerías de programación para el cálculo de entropía de Rényi y la aplicación de modelos de aprendizaje automático interpretables. Asimismo, se examina la situación actual de los ataques de baja tasa, que se caracterizan por su similitud con el tráfico legítimo, y se revisan trabajos previos que aplican técnicas de entropía y Machine Learning en entornos SDN.

3.2.1 Análisis de trabajos relacionados

Los cuatro trabajos seleccionados que se pueden observar en el Anexo 1 se tienden a considerar en el presente análisis porque representan los aportes más relevantes y actuales en la detección de ataques DDoS en redes SDN mediante el uso de entropía y técnicas de aprendizaje automático, cada uno desde un enfoque complementario que fundamenta y justifica la metodología propuesta en esta investigación. El trabajo de Puranik et al. (2023), por la introducción de un esquema híbrido de niveles dos en el que la entropía de Shannon realiza dicha función de filtro, siendo los de machine learning los que evidencian la anomalía, siendo los Random Forest una muestra de ello teniendo en cuenta que se puede obtener una reducción de falsos positivos. Fioravanti et al. (2023) ya realiza un estudio específico de la entropía en los mensajes de PACKET IN, mostrando como una alta concentración de tráfico hacia un solo destino provoca caídas drásticas en la entropía y la necesidad de ajustes en parámetros como el tamaño de ventana, el umbral al que se considera una anomalía y la persistencia de la misma para ser más precisos en la detección. El trabajo de Aladaileh y Anbar (2021) se incluye aquí porque usa entropía de Rényi y umbrales dinámicos y muestra mayores ventajas con respecto a aquel. Lo que permite la detección tanto de ataques en alta como de baja intensidad y que además apoya lógicamente el uso de Rényi como en esta tesis. Por último, el trabajo de Zhang et al. (2019) se ejecuta porque trata específicamente ataques de baja tasa (LDoS) mediante entropía basada en densidad espectral de potencia PSD-entropy, farming SVM y reducción de características, mostrando la utilidad de filtrados relativamente simples para muestras evidentes y modelos más complejos en muestras ambiguas y no evidentes. En conjunto, estos trabajos permiten identificar tendencias, limitaciones y oportunidades en la literatura, y proporcionan los fundamentos necesarios para plantear un

sistema de detección optimizado basado en entropía de Rényi y modelos de aprendizaje automático interpretables adecuados para escenarios SDN.

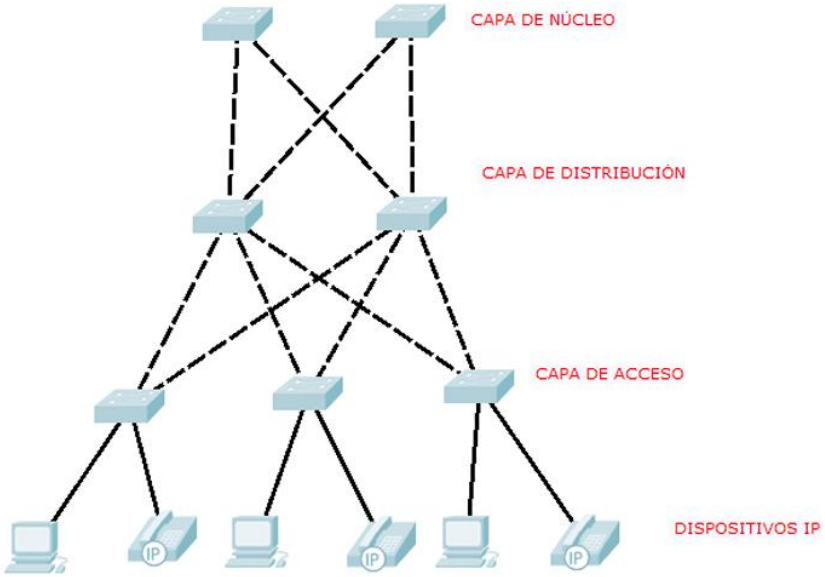
3.2.2 Análisis de la situación actual

El análisis de la situación actual constituye la primera fase dentro de la metodología en cascada, cuyo propósito es comprender de manera detallada el entorno técnico sobre el cual se desarrollará el sistema de detección de ataques DDoS de baja intensidad mediante análisis de entropía y modelos de aprendizaje automático interpretables. En esta etapa se revisan los componentes de la red existente, sus condiciones operativas, los flujos de información y los posibles puntos de vulnerabilidad que podrían ser aprovechados por un atacante.

La base estructural de la topología utilizada en esta investigación se fundamenta en el diseño presentado en la tesis de Edwin Paúl Tipán Calero, titulada “Rediseño de la infraestructura de red para la cooperativa de ahorro y crédito Pablo Muñoz Vega.”. Dicho trabajo propone una arquitectura jerárquica Cisco de tres capas (acceso, distribución y core), como se lo puede observar en la Figura 7, en donde se puede observar que se encuentra implementada en una empresa corporativa dedicada a finanzas. Este diseño es orientado a mejorar la disponibilidad, el rendimiento y la administración del tráfico dentro de la organización.

Figura 7

Diseño de red propuesta para la Cooperativa de Segmento 1



Nota: Obtenido de Edwin Tipán (2018)

La topología considerada para este proyecto se fundamenta en el diseño jerárquico propuesto en la tesis anteriormente mencionada, el cual organiza la red en tres capas: core, distribución y acceso. En este modelo, el núcleo se agrupa para las funciones de conmutación de alto rendimiento crítico, el enrutamiento inter-VLAN, la aplicación de políticas de seguridad y la salida hacia los enlaces WAN. La capa de distribución conecta las diferentes cabinas de comunicación mediante enlaces redundantes, como se ha visto en la Figura 7, garantizando una cierta continuidad del servicio en caso de fallas físicas, pero también controlando el flujo entre segmentos. Por último, la capa de acceso proporciona conectividad a los usuarios finales, dispositivos de videovigilancia, telefonía IP, impresoras, servidores u otros equipos corporativos, manteniendo la segmentación lógica como se ha definido a partir de las VLAN fijadas.

El diagnóstico detallado de la red incluyendo la topología mostrada y los segmentos identificados en su tesis además la evaluación del tráfico del switch de core evidencia que, si bien la infraestructura cuenta con una base sólida, existen limitaciones en segmentación, redundancia y monitoreo avanzado. Aunque la propuesta del documento introduce mejoras significativas a través de un diseño jerárquico y redundante, todavía no se dispone de mecanismos inteligentes que permitan identificar ataques. En respuesta a esta necesidad, el modelo expuesto en el presente trabajo se configura sobre la topología jerárquica introducida en la tesis, aportando una capa de seguridad de clase avanzada, tomando como base el análisis de entropía de flujos y modelos de aprendiza-je automático interpretables. Esto nos va a permitir anticiparse a la manifestación de anomalías en la red, detectar comportamientos fuera de lo normal dentro de la capa de control SDN y facilitar la disponibilidad de los servicios básicos, convirtiendo así a la topología en un entorno más elaborado y dispuesto a soportar ataques de baja tasa que pudieran comprometer poco a poco la continuidad de red corporativa real.

3.2.3 Requerimientos

La presente sección muestra los requisitos que son necesarios para alcanzar los objetivos que se plantea el proyecto, así mismo se detallan las consideraciones requeridas para una correcta implementación del sistema de detección de ataques de DDoS de baja intensidad para redes definidas por SDN. El objetivo es garantizar que el sistema pueda obtener los objetivos trabajados para el proyecto y que se adapte adecuadamente a la resolución de los requerimientos técnicos, funcionales y específicos de la seguridad considerados dentro del alcance del trabajo de investigación. Para el estudio, se realiza un análisis de los requisitos aplicables al proyecto, considerando la revisión bibliográfica y documental sobre detección de anomalías en tráfico de red basada en entropía de flujos y modelos de aprendizaje automático interpretables.

La ejecución y cumplimiento de este apartado se desarrollan conforme a la norma ISO/IEC/IEEE 29148:2018, la cual proporciona un marco estandarizado para la ingeniería de requerimientos, y al modelo System Requirements Architecture (SyRA), que permite organizar los elementos técnicos del sistema de manera estructurada.

A continuación, se presentan las categorías de requerimientos definidas por el estándar ISO/IEC/IEEE 29148:2018:

- **StSR**: Especificación de Requerimientos de las Partes Interesadas (Stakeholders).
- **SySR**: Especificación de Requerimientos del Sistema.
- **SRS**: Especificación de Requerimientos de Software.
- **SRH:** Especificación de Requerimientos de Hardware
- **MRS**: Especificación de Requerimientos del Modelo.
- **SyRA**: Especificación de Requerimientos de Arquitectura.

3.2.3.1 Requerimientos de Stakerholders (StSR)

Los stakeholders representan a las partes interesadas que intervienen directa o indirectamente en el desarrollo del sistema, incluyendo usuarios, responsables técnicos, patrocinadores y personal de gestión. Su participación es fundamental para identificar las necesidades, expectativas y restricciones que orientan el diseño del sistema y garantizan su coherencia con los objetivos estratégicos definidos en la investigación. En consideración a los principales aspectos del sistema, se especifican a las partes interesadas de la solución propuesta en la Tabla 6.

Tabla 6

Requerimientos de Stakeholders

Nro.	Stakeholders	Rol
StSR1	Ing. Fabián Cuzme	Tutor de tesis
StSR2	Ing. Edgar Maya	Asesor de Tesis
StSR3	Nahim Gómez	Estudiante

3.2.3.2 Requerimientos del Sistema (SySR). La norma ISO/IEC/IEEE 29148:2018 enfatiza la importancia de los requisitos en el proceso de diseño del sistema, pues cada elemento desempeña una función adecuada para conseguir cumplir con el cumplimiento de los objetivos generales del proyecto. Desde esa perspectiva, los requisitos del sistema constituyen la base de la definición de la arquitectura, de desarrollo de los módulos funcionales y de configuración de los procesos de sistema de detección**.**

Considerando estos lineamientos, la Tabla 7 presenta los requerimientos del sistema establecidos para el proyecto.

Tabla 7

Requerimientos del Sistema (SySR)

Requerimientos de la Interfaz					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SySR1	El sistema debe integrarse con el controlador SDN a través del protocolo OpenFlow para capturar flujos de tráfico.	X			
SySR2	Compatibilidad con módulos de monitoreo externos para importar o exportar métricas de red.		X		

Requerimientos de la Interfaz					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SySR3	Capacidad de interacción con sistemas de seguridad existentes.		X		
SySR4	El sistema deberá permitir la comunicación con el controlador SDN para enviar reglas de flujo que posibiliten la aplicación de medidas de mitigación	X			
SySR5	Generación de reportes por análisis del sistema mediante Logs			X	
SySR6	Visualización Gráfica de los resultados de clasificación detectados.		X		
Requerimientos de Uso					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SySR7	El sistema debe permitir ajustar parámetros operativos como el intervalo de muestreo, la ventana de análisis y el umbral de entropía.		X		
SySR8	El sistema debe exportar resultados de la clasificación y métricas de rendimiento		X		SySR2
SySR9	El sistema deberá operar de forma autónoma, detectando de manera continua patrones de tráfico anómalo y generando alertas automáticas que notifiquen al controlador para su correspondiente mitigación.	X			
Requerimientos de Perfomance					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SySR10	Alta precisión en la detección de ataques DDoS de baja intensidad		X		

Requerimientos de la Interfaz					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SySR11	Uso eficiente de los recursos del sistema (CPU, RAM) durante la recolección, análisis y clasificación del tráfico		X		
SySR12	Capacidad de procesamiento continuo de flujos sin degradar el rendimiento	X			
SySR13	Tiempo de respuesta de detección y mitigación	X			
Requerimientos de modo y estado					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SySR14	El sistema operará con un solo controlador.	X			
SySR15	Capacidad para reiniciar procesos y restaurar configuraciones sin pérdida de datos tras una interrupción.		X		
SySR16	Capacidad del Sistema para registrar los cambios entre modo normal y modo detección durante las pruebas.			X	

3.2.3.3 Requerimientos de Software (SRS). El parámetro SRS describe los requerimientos tanto funcionales como los no funcionales que deben cumplir el software que se va a desarrollar, detallando aspectos de consideración como el propósito del sistema, sus principales características, el nivel de rendimiento esperado y su comportamiento general. Por lo cual se establecen los siguientes requerimientos de ponderación presentados en la Tabla 8**.**

Tabla 8

Requerimientos de Software (SRS)

Requerimientos de Software					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SRS1	El tipo de licencia del software del controlador debe ser de código abierto.	X			
SRS2	El software debe ser Compatible con Sistemas Operativos como Linux.		X		
SRS3	El lenguaje de programación debe tener compatibilidad con librerías de análisis de datos y aprendizaje automático (ML).	X			
SRS4	El IDE de programación debe ser de código abierto.	X			
SRS5	El software de simulación debe tener la capacidad para diseñar, gestionar y simular topologías de red	X			
SRS6	El software de simulación debe ser capaz de usar los recursos (dispositivos, servicios y controlador SDN)	X			

3.2.3.4 Requerimientos de Hardware (SRH). El parámetro SRH describe los requerimientos y componentes de hardware necesarios para garantizar el correcto desarrollo del entorno de simulación empleado en la presente investigación. Estos requerimientos se definen considerando la ejecución de una red SDN simulada, la generación de tráfico, el desarrollo del sistema de detección con el modelo de aprendizaje automático interpretable. Por lo cual se establecen los siguientes requerimientos mostrados en la siguiente Tabla 9.

Tabla 9

Requerimientos de Hardware (SRH)

Requerimientos de Hardware					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SRH1	El equipo deberá contar con una capacidad mínima de 24 GB de memoria RAM para soportar la ejecución simultánea del proyecto.	X			
SRH2	El equipo deberá disponer de un procesador de gama baja-media que permita la ejecución paralela de las máquinas virtuales y procesos del sistema de detección.		X		
SRH3	El proyecto puede operar en cualquier sistema operativo como Linux o Windows.		X		
SRH4	El equipo puede tener disponible una tarjeta gráfica que permita acelerar la fase de entrenamiento del modelo.			X	
SRH5	Tener suficiente capacidad de almacenamiento (mínimo 512 GB)		X		

3.2.3.5 Especificación de Requerimientos del Modelo (MRS). Los modelos Machine Learning tienen una importancia esencial, ya que determinan el grado de eficiencia, precisión y adaptabilidad del sistema. La elección del modelo influye directamente en la capacidad para identificar patrones, procesar grandes volúmenes de datos y generar resultados confiables. Por esta razón, resulta necesario establecer una especificación clara de los requisitos del modelo Machine Learning en la cual se llegan a detallar en la Tabla 10.

Tabla 10

Requerimientos del modelo

Requerimientos del Modelo					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
MRS1	Capacidad para gestionar y procesar grandes volúmenes de datos	X			
MRS2	El tipo de licencia del modelo ML debe ser OpenSource	X			SRS1
MRS3	Baja Complejidad de implementación Computacional		X		
MRS4	Bajo consumo de recursos del equipo para el entrenamiento del modelo			X	
MRS5	Facilidad de interpretación		X		
MRS6	Capacidad para ajustar y optimizar parámetros	X			
MRS7	Compatibilidad con modelos interpretables	X			

3.2.3.6 Especificación de Requerimientos de Arquitectura (SyRA). La especificación de requerimientos de arquitectura establece los componentes estructurales y las relaciones que conforman el sistema, asegurando la coherencia entre los distintos módulos y su correcta integración en el entorno de desarrollo. Este nivel de definición permite representar la distribución lógica y funcional del sistema, así como los flujos de información que se generan entre sus elementos.

Es por eso por lo que en la Tabla 11 se establecen los siguientes requisitos que servirán para el desarrollo del sistema.

Tabla 11

Requerimientos de Arquitectura

Requerimientos de la Arquitectura del Sistema					
#	Descripción	Prioridad			Relacionado con
		Alta	Media	Baja	
SyRA1	La arquitectura del sistema debe representar una estructura jerárquica de red que contemple niveles de acceso, distribución y núcleo.	X			
SyRA2	El sistema de detección debe mantener interconexión estable y comunicación constante entre los dispositivos de red.	X			SySR4
SyRA3	La arquitectura debe ser escalable, permitiendo añadir nuevos nodos en la topología sin afectar la estabilidad general.	X			SRS5
SyRA4	El entorno de simulación debe cumplir con los aspectos básicos incluyendo conectividad, procesamiento y transmisión.		X		SRS5
SyRA5	El modelo debe clasificar el ataque de acuerdo con lo configurado.	X			

3.2.4 Selección de Software

En esta sección se realizará la selección de los componentes fundamentales que conforman la propuesta técnica del sistema de detección, con el propósito de garantizar coherencia entre el diseño planteado y su posterior implementación. Con esa finalidad, se procederá al análisis, de forma estructurada, de los siguientes elementos: el software de simulación, el modelo de detección,

el modelo interpretable, el controlador SDN, el lenguaje de programación y el entorno de desarrollo, considerando en cada caso su funcionalidad, su compatibilidad con la arquitectura propuesta para la solución y su aportación al cumplimiento de los objetivos del proyecto, ya que, debido a esta selección, se podrá establecer una base tecnológica suficiente para llevar a cabo el despliegue del sistema en un entorno SDN simulado. orientado a la detección de ataques DDoS de baja intensidad.

3.2.4.1 Selección del Software de Simulación. Este apartado se contextualiza entorno a la herramienta de simulación y desarrollo, cuya selección se fundamenta en criterios de compatibilidad, flexibilidad y capacidad para integrar técnicas basadas en aprendizaje automático. La elección del software resulta determinante para garantizar la correcta implementación del modelo propuesto.

Por consiguiente, la selección del entorno de simulación se establece como un elemento clave para la validación del modelo, permitiendo recrear de forma controlada los escenarios necesarios para el análisis y la evaluación de los resultados. De esta manera, se lleva a cabo un análisis comparativo entre las principales herramientas disponibles, considerando aspectos anteriormente mencionados.

La Tabla 12 presenta los criterios de ponderación aplicados para la selección del software, tomando como referencia los requerimientos establecidos en la Tabla 9, correspondiente a los requisitos del software, y tomando en cuenta sus debidas características mostradas en el Anexo 1.

Tabla 12

Selección del Software de Simulación

Software de Simulación	Requisitos						Total
	SRS1	SRS2	SRS3	SRS4	SRS5	SRS6	
NS-3	1	1	0	0	1	1	3
OMNeT++	1	1	0	1	1	0	4
GNS3	1	1	1	1	1	1	6
Mininet	1	1	0	1	1	1	5
1 = Cumple, 0 = No cumple							

De acuerdo con los resultados obtenidos en la Tabla 12, se selecciona GNS3 como el software de simulación más adecuado para el desarrollo del sistema de detección, al cumplir con la totalidad de los requerimientos establecidos. Este entorno ofrece una plataforma versátil para el diseño, gestión y prueba de arquitecturas de red definidas por software (SDN), garantizando la compatibilidad con el controlador seleccionado y con los módulos de análisis implementados en lenguaje de programación de código abierto.

3.2.4.2 Selección del Modelo. Para fundamentar la selección del modelo de aprendizaje automático, se realizó el análisis de requerimientos anteriormente mencionados. De esta manera, se establece una base comparativa sólida que permite seleccionar el modelo más adecuado de acuerdo con los objetivos planteados y las condiciones del sistema propuesto. El proceso de selección considera la precisión, interpretabilidad, capacidad de generalización y eficiencia computacional de los modelos revisados, garantizando su idoneidad para la detección de ataques de baja intensidad.

En la Tabla 13 se presenta los principales criterios de ponderación para la selección del modelo la cual es referenciado en la Tabla 10 correspondiente a los requerimientos del modelo y tomando en cuenta las características de cada modelo mostradas.

Tabla 13

Selección del Modelo

Modelos	Requisitos						Total
	MRS1	MRS2	MRS3	MRS4	MRS5	MRS6	
SVM	0	1	1	0	0	0	2
Random Forest	1	1	1	1	1	0	5
XGBoost	1	1	1	1	1	1	6
K-Means	0	1	0	1	0	0	2
1 = Cumple, 0 = No cumple							

Según los resultados obtenidos en la Tabla 13, el modelo XGBoost es el que cumple con la totalidad de los requisitos establecidos en la Tabla 10, destacando por su capacidad de gestionar grandes volúmenes de datos, su eficiencia operativa, y su compatibilidad con modelos interpretables, lo que permite explicar el comportamiento de sus predicciones. De esta manera, se escoge XGBoost como el modelo de aprendizaje de máquinas que más se acomoda al sistema de detección, resultando de dicha combinación, un modelo que si bien es efectivo y preciso, resulta de fácil interpretación para los humanos, algo que consideramos un requisito deseado de un modelo a utilizar en un sistema de detección.

3.2.4.3 Selección del Modelo Interpretable. La interpretabilidad del modelo representa parte fundamental de la estructura de detección, puesto que sirve para entender de forma clara cómo el clasificador toma decisiones en los momentos de identificar tráfico legítimo o anómalo. En un entorno de seguridad, donde las acciones de mitigación deben estar justificadas y ser auditables, resulta indispensable incorporar un mecanismo que permita explicar el aporte de cada característica del tráfico en el proceso de clasificación. Para ello, se evaluaron distintos métodos de interpretabilidad con base en los requerimientos del modelo establecidos en la Tabla 10.

Con estos criterios se analizaron cuatro enfoques utilizados en la explicación de modelos de aprendizaje automático que se pueden visualizar mediante la Tabla 14.

Tabla 14

Selección del Modelo Interpretable

Método	Requisitos						Total
	MRS1	MRS2	MRS3	MRS4	MRS5	MRS6	
SHAP	1	1	1	1	1	1	6
LIME	1	1	1	0	1	0	4
Permutation Importance	0	1	0	1	0	1	3
Feature Importance	0	1	0	1	0	0	2
1 = Cumple, 0 = No cumple							

Los resultados evidencian que SHAP (SHapley Additive exPlanations) es el método que cumple la totalidad de los criterios evaluados. SHAP, fundamentado en la teoría de estos valores de Shapley, ofrece explicaciones coherentes y justificadas de un modelo a nivel local y global. A

su vez, SHAP es 100% compatible con algoritmos de ensambles y boosting, ofreciendo interpretabilidad sin pérdida de rendimiento del clasificador.

3.2.4.4 Selección del Controlador. La elección del controlado es el elemento más crítico en el diseño de la arquitectura SDN, ya que hace referencia a quién debe gestionar el plano de control, cómo se establece la comunicación con los dispositivos de red y cómo se implementan las políticas de detección y mitigación que han sido configuradas en el sistema. La elección del controlador influye directamente en el rendimiento, la latencia y la capacidad de integración con los módulos de análisis del tráfico y los mecanismos de respuesta automática.

Para su selección, se consideraron los requerimientos más relevantes establecidos de la Tabla 8 y Tabla 9 en la cual las podemos observar en la siguiente Tabla 15 teniendo en cuenta las características de cada controlador.

Tabla 15

Selección del Controlador

Controladores	Requisitos							Total
	SySR1	SySR4	SySR11	SySR12	SRS1	SRS2	SRS3	
Ryu	1	1	1	0	1	1	1	6
OpenDaylight	1	1	0	1	1	1	0	5
Floodlight	1	1	0	0	1	1	0	4
POX	1	1	0	0	1	1	1	5
1 = Cumple, 0 = No cumple								

La Tabla 15 presenta la comparación de los principales controladores SDN en función de los requerimientos establecidos. En relación con la evaluación de los resultados, se ha decidido optar por Ryu, dada su conformidad respecto a la gran mayoría de los criterios establecidos y que,

además, presenta una arquitectura, en este caso, ligera, modular y que cumple con la necesidad de ser compatible con OpenFlow de la misma manera que los módulos de detección y de respuesta. La elección del Python como lenguaje de desarrollo de Ryu propicia la posibilidad de una conexión directa con el sistema para la detección de anomalías basado en entropía y en aprendizaje automático, posibilitando por tanto una conexión muy natural con los módulos de análisis.

3.2.4.5 Selección del Lenguaje de programación.

La elección del lenguaje de programación puede considerarse dentro de las variables para el diseño del mismo, dado que condiciona cómo los módulos y las herramientas desarrolladas/deben de ser implementadas para la ejecución del proyecto de investigación. La elección del lenguaje influye mucho para el diseño del entrenamiento del modelo, tratamiento de datos, rendimiento y configuración del controlador del sistema además de la visualización grafica de los resultados. Para su selección, se consideraron los requerimientos establecidos en la Tabla 8, Tabla 9 y Tabla 10. En base a ellos, se evaluaron 4 lenguajes de programación que se detallan en la siguiente Tabla 16 para su comparación.

Tabla 16

Selección del Lenguaje de Programación

Lenguajes	Requisitos							Total
	SySR1	SySR4	SySR7	SySR11	MRS1	MRS3	MRS6	
Python	1	1	1	1	1	1	1	7
C++	1	0	1	1	0	1	0	4
Java	1	1	1	1	1	0	0	5
JavaScript	0	0	0	1	1	0	0	2

1 = Cumple, 0 = No cumple

Como se pudo observar en la Tabla 16 se evidencia que Python cumple con la mayor cantidad de requerimientos relevantes establecidos anteriormente como la compatibilidad con las librerías, cálculos estadísticos importantes y configuración del sistema con el modelo planteado. Dado lo anterior, se selecciona Python como lenguaje principal del sistema, al aportar mayor flexibilidad, rapidez de desarrollo y cohesión con los módulos de entrenamiento, controlador y clasificación.

3.2.4.6 Selección del Entorno de Programación. La elección del entorno de programación es un componente determinante dentro del proceso de diseño y desarrollo del sistema, ya que define las herramientas con las que se construyen, prueban y depuran los módulos que conforman la solución. Un entorno adecuado debe facilitar la integración de librerías, permitir la gestión eficiente del código, ofrecer opciones avanzadas de depuración y asegurar compatibilidad con los requerimientos tanto del lenguaje seleccionado como del sistema en general. Con base en estos lineamientos, se evaluaron cuatro entornos de programación que se muestran en la siguiente Tabla 17 ofreciendo características relevantes para los requerimientos de la investigación, comparándolas eficientemente para su debida elección.

Tabla 17

Selección del Entorno de Programación

IDE	Requisitos						Total
	SRS2	SRS3	SRS4	MRS5	SySR10	SyRA4	
VS Code	1	1	1	1	1	1	6
JupyterLab	1	1	1	0	1	1	5
Eclipse	1	0	1	0	0	1	3
PyCharm	1	1	0	1	1	1	5
1 = Cumple, 0 = No cumple							

En referencia con la Tabla 17 se muestra que VS Code obtiene la mayor puntuación debido a que cumple con la totalidad de requerimientos evaluados como la compatibilidad con el lenguaje de programación, ser de código abierto y además de integrar módulos e integración para el entorno de simulación escogido anteriormente.

Por estas razones, se ha optado por utilizar VS Code como el principal entorno para llevar a cabo el proceso de desarrollo del sistema, ya que permite centralizar en un solo espacio la programación del controlador, los módulos para el entrenamiento y el módulo de visualización gráfica de los resultados.

3.3 Diseño

En cuanto al diseño se puede mencionar que se lo desarrolló a partir de un conjunto de fases estructuradas que permiten organizar de forma lógica y progresiva los elementos que se requieren para la construcción del sistema de detección. Cada fase cumple una función específica dentro del proceso general, garantizando coherencia entre la arquitectura propuesta, las herramientas seleccionadas y los objetivos funcionales del sistema.

Como parte inicial se desarrolla el diseño de la topología de red, basada en el modelo jerárquico de tres niveles. En esta etapa se definen las capas de acceso, distribución y núcleo, así como el direccionamiento y los dispositivos que conformarán la red SDN. La topología presentada servirá como banco de pruebas para el funcionamiento de los diferentes escenarios experimentales planteados a lo largo de la investigación.

Finalmente, se afronta el diseño del controlador y de los componentes analíticos del sistema, que incluyen el diseño del control plane, el proceso del diseño del dataset y la integración del sistema de detección. Esta integración incluye la incorporación del módulo de entropía de Rényi y del modelo interpretable previamente creado, permitiendo establecer el flujo completo de análisis, clasificación y respuesta en el contexto SDN.

3.3.1 Diseño de la topología

El diseño de la topología de red SDN propuesta en la Figura 8 se desarrolló tomando como referencia los principios jerárquicos empleados en el trabajo de Reyes (2022), titulado *"Rediseño de la infraestructura de red para la cooperativa de ahorro y crédito Pablo Muñoz Vega"*, mostrado en la Figura 7. Aunque dicho estudio no especifica explícitamente el nivel SEPS de la institución, la descripción operativa presentada cuenta con más de 50.000 socios, muchas agencias y una infraestructura que requiere alta disponibilidad es propio de una cooperativa del Segmento 1, según los criterios de clasificación que ofrece la Superintendencia de Economía Popular y Solidaria, que le otorga el rasgo distintivo de manejar una notable capacidad de tráfico, de utilizar simultáneamente muchos servicios y de requerir una arquitectura de red que exige estabilidad, escalabilidad y segmentación suficiente, características que permiten justificar la adopción de un modelo jerárquico de tres capas.

Bajo esta premisa, el trabajo de Reyes (2022) sirvió como base conceptual para estructurar la arquitectura jerárquica implementada en esta propuesta, adaptándola a un entorno definido por software (SDN) orientado a la detección de ataques DDoS de baja intensidad. La topología desarrollada replica la separación funcional entre acceso, distribución y núcleo propia de redes financieras de segmento 1, permitiendo un control eficiente del tráfico, una gestión ordenada de los dispositivos y una respuesta rápida ante cambios en el comportamiento de la red. En la siguiente Tabla 19 se presenta la verificación del cumplimiento de los requerimientos de arquitectura en relación con la topología diseñada y su correspondencia con la Figura 8 que ilustra la distribución física y lógica de los componentes del sistema SDN propuesto.

Tabla 18

Cumplimiento de Requerimientos de arquitectura

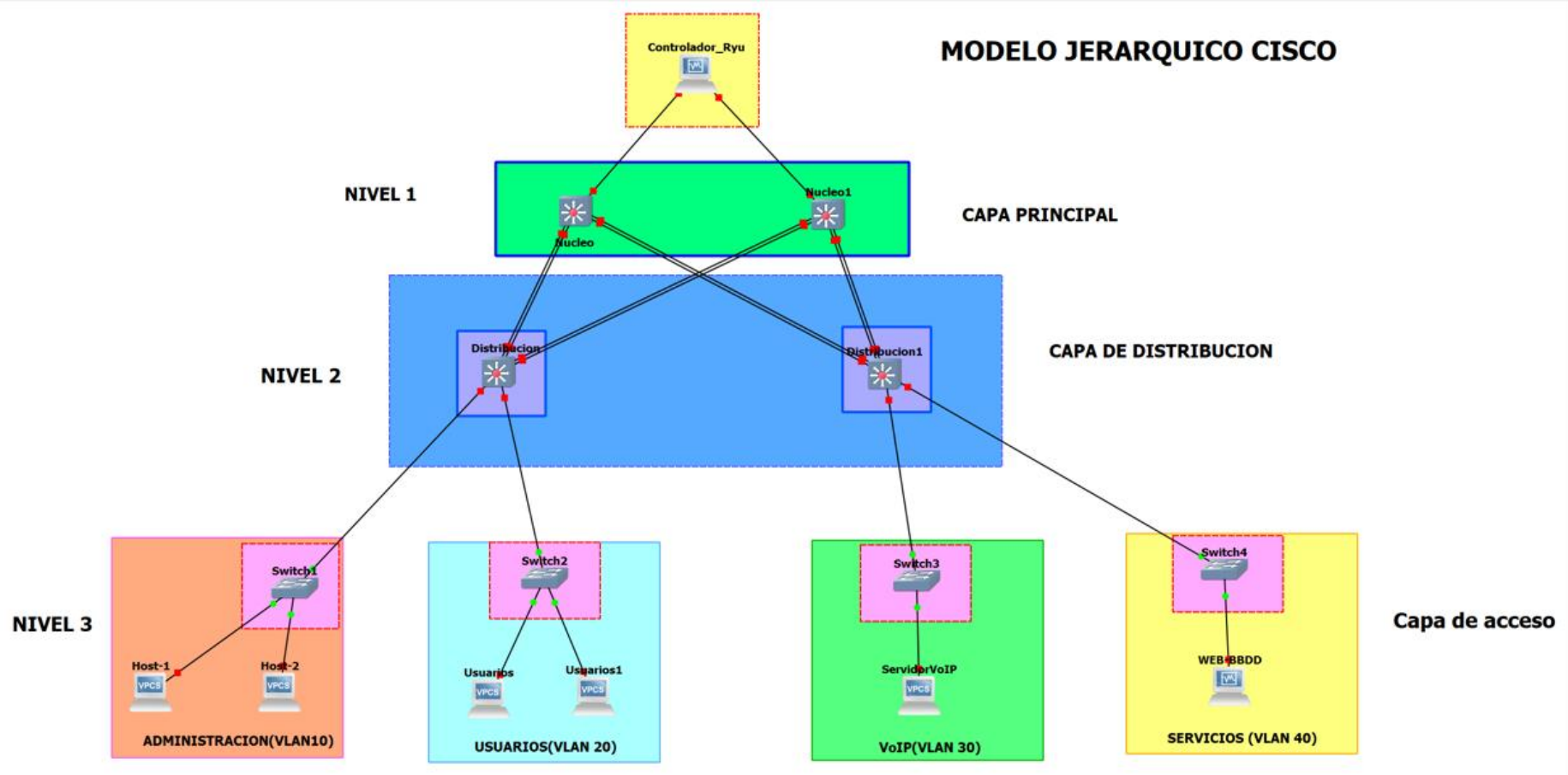
Requerimientos	**Cumplimiento**		
	Alta	**Media**	**Baja**
SyRA1	X		
SyRA2	X		
SyRA3	X		
SyRA4			X
SyRA5	X		

En la siguiente Figura 8 se muestra la topología diseñada, la cual integra las capas de acceso, distribución y núcleo, complementadas con un controlador Ryu que centraliza el plano de control y coordina la interacción con los módulos de detección basados en entropía de Rényi y modelos de aprendizaje automático interpretables. Esta configuración jerárquica, alineada con la

estructura tradicional utilizada en instituciones financieras de Nivel 1, garantiza la conectividad estable entre los dispositivos, facilita la recolección continua de métricas de tráfico y permite la ejecución dinámica de reglas de mitigación desde el controlador hacia los switches OVS ubicados en cada nivel funcional de la arquitectura.

Figura 8

Topología de red SDN



3.3.1.1 Dispositivos que conforman la Red. Una vez que se definió el escenario general de esta investigación y la ubicación de los dispositivos de acuerdo con el diseño de la topología de la Figura 8, este apartado se enfocara en mencionar la información de cada uno de los dispositivos que conforman la red tanto como hosts, dispositivos de red, y servidores, teniendo como mayor punto importante la especificación de los recursos del sistema que estará en el entorno GNS3. La siguiente Tabla 19 se muestra de manera detallada la información de uso como memoria RAM Y CPU, teniendo en cuenta las características de hardware como host anfitrión y por ultimo los sistemas operativos utilizados para cada dispositivo y su función en la red SDN.

Tabla 19

Dispositivos que conforman la topología.

Dispositivo	RAM	CPU	Sistema Operativo	Rol en la Red
Controlador	4 GB	3	Ubuntu 20.04	Controlador SDN
Detección Externa	4 GB	3	Ubuntu 20.04	Detección Externa
Switches	512 MB	1	OpenVSwitch	Switches SDN
VoIP	2 GB	1	Ubuntu 20.04	Servidor Voz IP
Base de Datos y Web	2 GB	1	Ubuntu 20.04	Servicios
Usuarios	32 GB	12	Arch Linux	Bots Usuarios
Atacantes	32 GB	12	Arch Linux	Bots Atacantes

3.3.1.2 Direccionamiento IPv4 de la topología. Una vez que establecemos la distribución de los dispositivos y la topología adecuada que conforman al escenario de pruebas, la siguiente fase corresponde a la asignación del direccionamiento IP para la red correspondiente teniendo dentro tanto como, conmutadores SDN, Hosts, Servidores y el módulo de detección. La siguiente Tabla 20 se describe a detalle el nombre del dispositivo, su interfaz, dirección IP, máscara de red y la dirección de la subred. Se ha tomado como referencia la IP 192.168.56.0/24, la cual para el cálculo aplicaremos VLSM para así poder determinar las máscaras y direccionamiento de toda la red correspondiente.

Tabla 20

Direccionamiento IPv4 de la topología de red

Dispositivo	Interfaz	Dirección IP	Gateway	Mascara de Subred
OVS-Core 1	Br01	192.168.56.1	--	255.255.255.0
OVS-Core 2	Br02	192.168.56.2	--	255.255.255.0
OVS-Distribución 1	Br-dist1	192.168.56.3	--	255.255.255.0
OVS-Distribución 2	Br-dist2	192.168.56.4	--	255.255.255.0
Controlador	Eth0, Eth1, Eth2	192.168.56.5, 192.168.56.6,192.168.57.60	--	255.255.255.0
Detección Externa	Eth0	192.168.57.70	--	255.255.255.0
BBDD_Web	Eth0	192.168.56.20	--	255.255.255.0
VoIP	Eth0	192.168.56.25	--	255.255.255.0

Dispositivo	Interfaz	Dirección IP	Gateway	Mascara de Subred
Usuarios	Virbr1	192.168.56.110	--	255.255.255.0
Atacantes	Virbr0	192.168.56.100	--	255.255.255.0

3.3.2 Diseño de la arquitectura del controlador y monitoreo

3.3.2.1 Arquitectura del controlador Ryu. La puesta en marcha de una red SDN empieza fundamentalmente en el plano de control, puesto que en dicho componente se engloban todas las funciones de gestión, control y coordinación de la propia red. Es aquí donde reside la inteligencia que interpreta en tiempo real el estado de la propia infraestructura, toma decisiones sobre el enrutado del tráfico, gestiona el modo de operar de los dispositivos del plano de datos. A diferencia de las arquitecturas tradicionales, donde cada equipo físico implementa su propio mecanismo de control, en SDN toda esta funcionalidad se centraliza en un único controlador de red, lo que permite conseguir una mayor flexibilidad a la hora de operar y a la vez una visión general de la propia red.

Los controladores SDN se implementan por lo general mediante plataformas de software de código abierto que funcionan bajo los frameworks específicamente concebidos para el control programado de la red, que funcionan mediante lenguajes de programación extensibles y que incorporan librerías de gestión de tráfico, procesamiento de eventos y comunicación con los switches. De manera general, se distribuyen bajo licencias abiertas que facilitan el desarrollo de aplicaciones, la ampliación de funcionalidades y la interoperabilidad con soluciones de terceras partes.

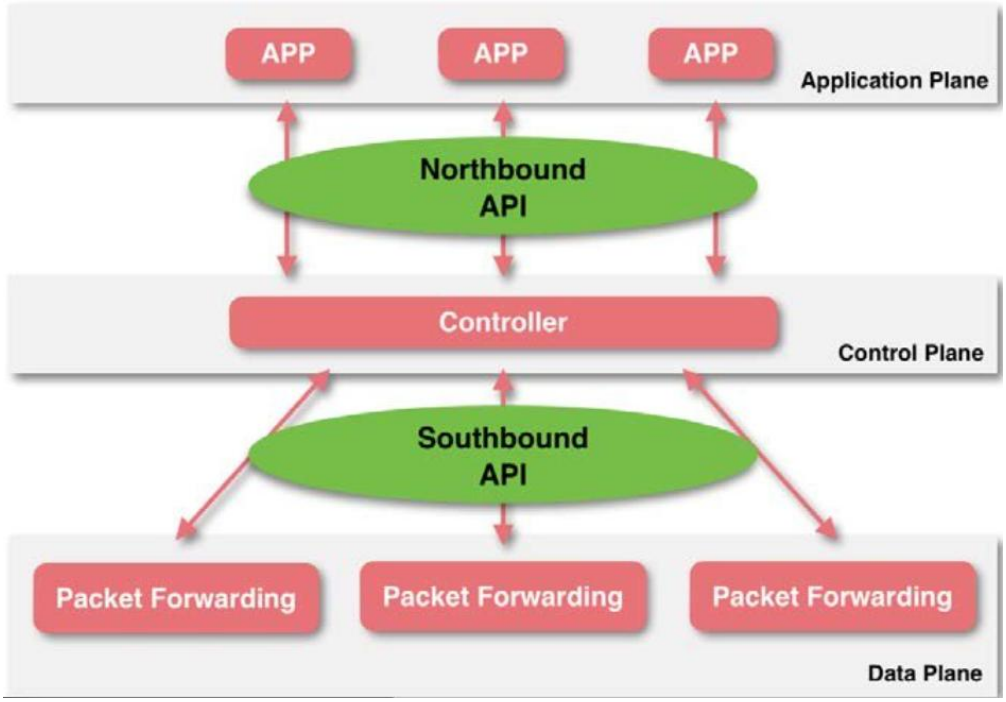
El controlador SDN es compatible con protocolos que permiten la comunicación con los dispositivos del plano de datos. El estándar más utilizado es OpenFlow, el cual define los mensajes

que se intercambian entre el controlador y los conmutadores para instalar reglas de flujo, consultar estadísticas o notificar eventos. En este sentido, un controlador SDN moderno debe ser capaz de:

- Analizar los paquetes entrantes reportados por los switches.
- Generar y enviar mensajes de control para gestionar la red.
- Organizar bibliotecas internas que respalden las operaciones de procesamiento del tráfico.
- Integrar aplicaciones adicionales, tales como módulos de monitoreo, funciones de enrutamiento, cortafuegos o sistemas de detección.

Figura 9

Arquitectura de un Controlador SDN



Nota: Obtenido de Jianqing Wu (2015)

La Figura 9 muestra la arquitectura de un controlador SDN, resaltando su compatibilidad con interfaces northbound para comunicarse con aplicaciones externas y southbound para

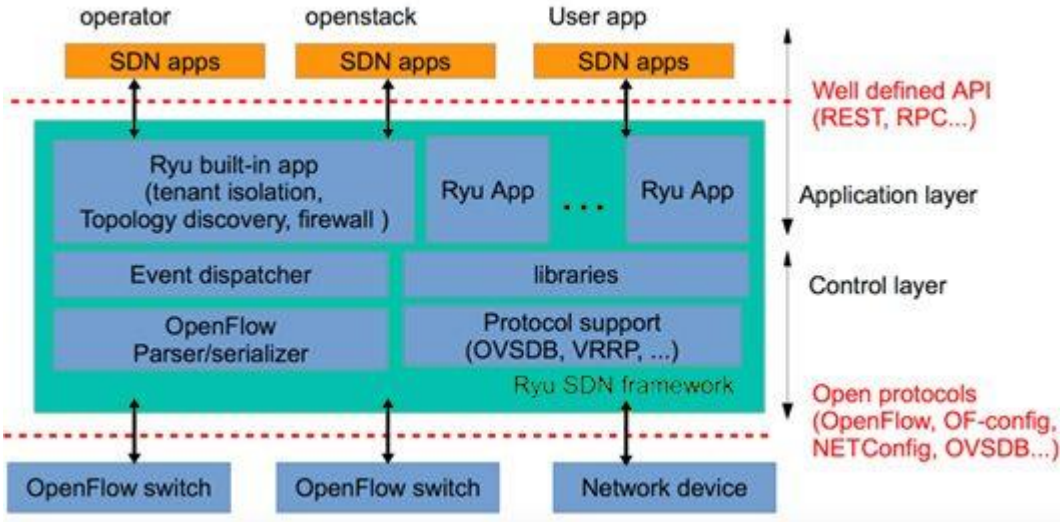
programar dispositivos de red a través de OpenFlow. La arquitectura contempla un gestor principal responsable de iniciar, ejecutar y coordinar las aplicaciones internas, garantizando que los flujos definidos por el usuario se apliquen en los conmutadores de acuerdo con las necesidades operativas de la red.

La arquitectura del controlador SDN establece que el funcionamiento de la red se basa en eventos, lo que significa que las aplicaciones alojadas en el plano de control reaccionan a sucesos específicos que ocurren en la infraestructura. Por ejemplo, cuando un conmutador recibe un paquete para el cual no existe una regla de flujo, genera un evento que es enviado al controlador. En respuesta, las aplicaciones encargadas del enrutamiento o la seguridad pueden decidir instalar nuevas reglas, reenviar el tráfico o aplicar políticas específicas.

Cada aplicación ejecutada en el controlador funciona como una entidad independiente, gestionando su propio conjunto de eventos. Estas aplicaciones se organizan con colas internas que reciben los mensajes provenientes de la red de forma ordenada. En la siguiente Figura 10 se muestra la arquitectura para el controlador Ryu de una aplicación ejecutada dentro del controlador SDN, en la cual cada aplicación utiliza un hilo dedicado para procesar eventos en su cola. El hilo extrae los eventos, los analiza y ejecuta las acciones correspondientes según su lógica interna.

Figura 10

Arquitectura del Controlador Ryu



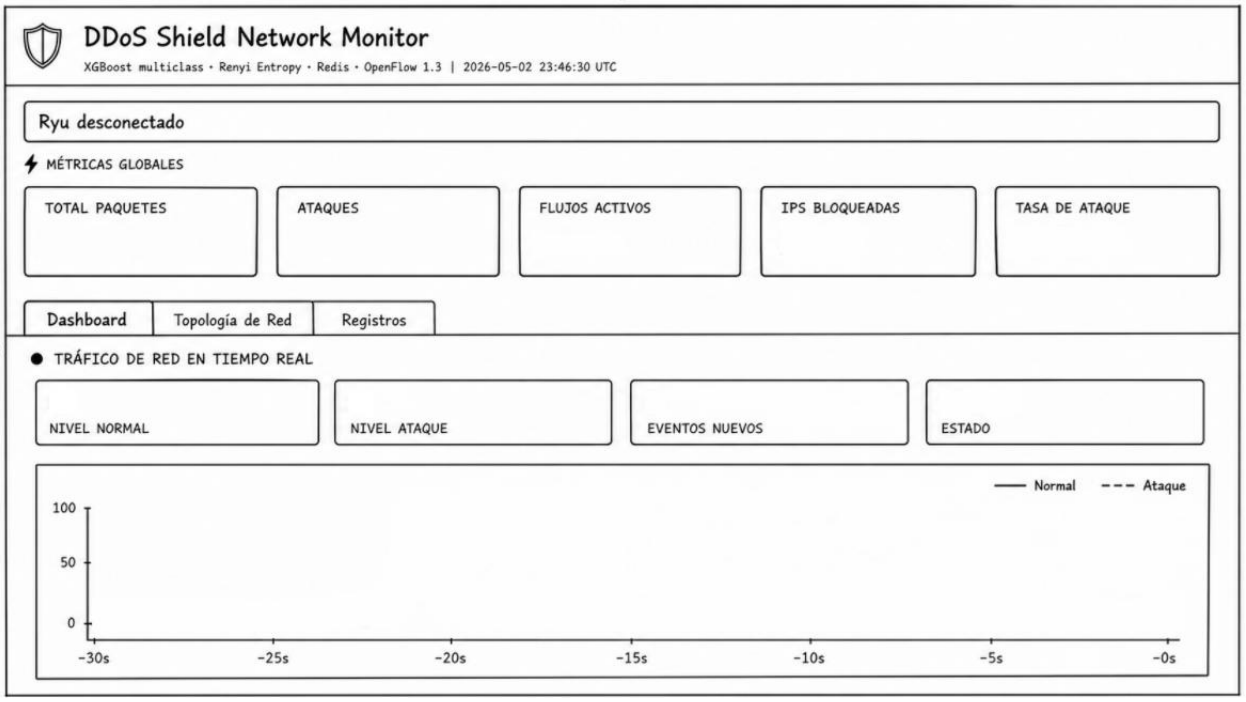
Finalmente, las aplicaciones alojadas en el controlador se gestionan a través de herramientas internas proporcionadas por el propio framework del controlador. Estas herramientas permiten iniciar aplicaciones, detenerlas, supervisarlas y listar todas las funciones disponibles dentro del entorno de control. Su propósito es ofrecer una plataforma ordenada y modular que permita a los desarrolladores implementar, ejecutar y mantener aplicaciones que gobiernen el comportamiento de la red SDN.

3.3.2.2 Diseño del Dashboard de visualización del controlador. El dashboard de visualización del controlador se concibe como una interfaz de monitoreo centralizada, orientada a representar el estado operativo del sistema de detección y mitigación de ataques DDoS implementado sobre el controlador Ryu. La propuesta de diseño del panel responde a la necesidad de disponer de una capa de observabilidad que permita interpretar el comportamiento de la red SDN en tiempo real, combinando métricas globales, el resultado del modelo de clasificación, valores de entropía, tiempos de detección, sucesos de tráfico y logs de bloqueo. De esta manera, el panel no se limita a mostrar datos por separado, sino que centraliza la información del sistema en pestañas operativas que favorecen la supervisión, el análisis y la validación del mecanismo de defensa que se propone.

Como se observa en la siguiente Figura 11 la primera parte de la pestaña del Dashboard presenta la interfaz principal denominada DDoS Shield Network Monitor. En el encabezado se identifican los componentes técnicos asociados al funcionamiento del sistema, tales como el modelo XGBoost multiclase, la entropía de Rényi, el uso de Redis y el protocolo OpenFlow 1.3. Además, se muestra el estado de conexión del controlador Ryu, lo cual permite verificar si el sistema se encuentra enlazado correctamente con el plano de control. Esta información inicial resulta importante porque permite al operador conocer, desde el primer nivel de visualización, si el entorno de monitoreo está activo y en condiciones de procesar eventos de red.

Figura 11

Diseño del Dashboard del controlador - primera pestaña

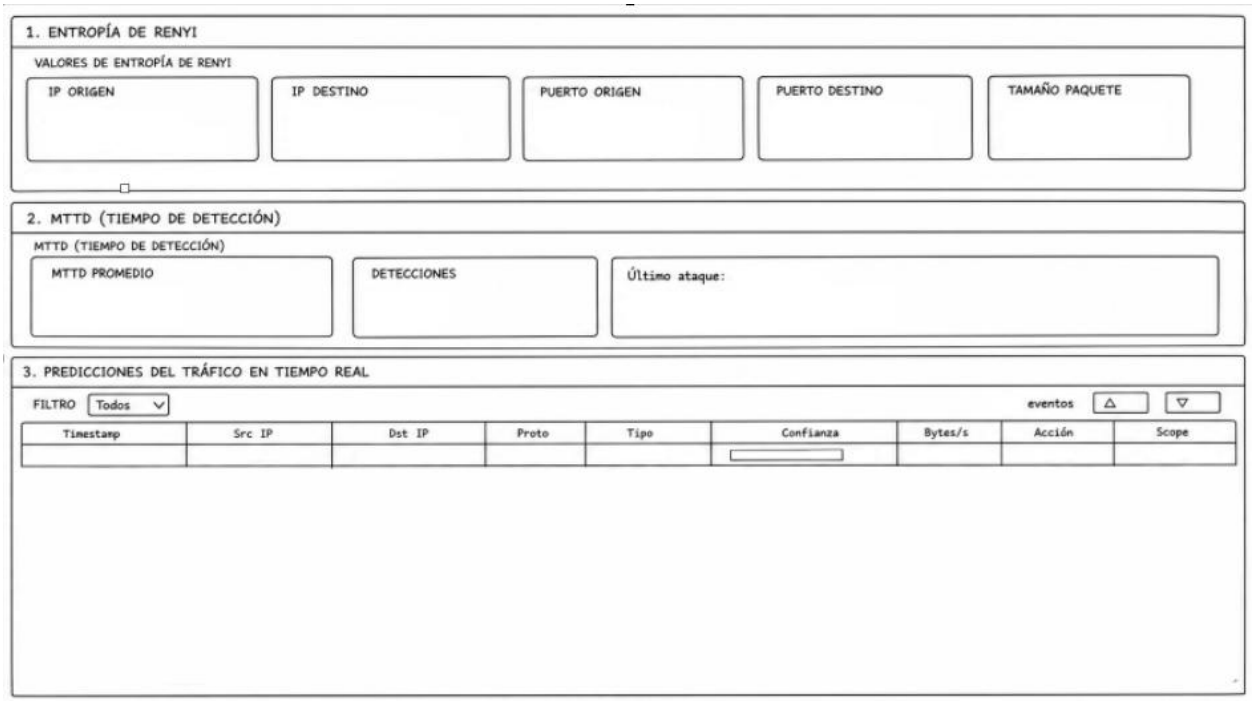


En esta misma sección se incorporan las métricas globales del sistema, entre las cuales se consideran el total de paquetes procesados, el número de ataques detectados, los flujos activos, las direcciones IP bloqueadas y la tasa de ataque. La información proporcionada por esos indicadores, que presenta una visión resumida del comportamiento general de la red, permite determinar rápidamente si el tráfico se encuentra dentro de estas condiciones normales o si, en cambio, empiezan a surgir síntomas que puedan asociarse a un posible ataque. Adicionalmente, la sección de tráfico de red en tiempo real recoge tanto los indicadores de nivel normal como de nivel de ataque, junto con los nuevos eventos y con el estado, junto con la gráfica temporal que presenta la diferencia en el comportamiento entre el normal y aquel clasificado como ataque. Esta organización facilita la supervisión continua del entorno SDN y permite observar la evolución del tráfico en una ventana de tiempo reciente.

Por otro lado, también se tiene presente otra característica como se aprecia en la Figura 12, la segunda parte de la pestaña Dashboard profundiza en los indicadores técnicos utilizados para el análisis del tráfico. En primer lugar, se presenta la sección de entropía de Rényi, en la cual se visualizan valores asociados a la IP de origen, IP de destino, puerto de origen, puerto de destino y tamaño del paquete. La incorporación de estos elementos permite observar variaciones en la diversidad y distribución del tráfico, lo cual resulta relevante para identificar comportamientos anómalos compatibles con ataques DDoS. Desde el punto de vista del diseño, esta sección transforma una métrica matemática compleja en indicadores visuales interpretables para el monitoreo operativo del sistema.

Figura 12

Diseño del Dashboard del controlador – pestaña Indicadores técnicos



De forma complementaria, esta misma vista incorpora la sección de MTTD, correspondiente al tiempo medio de detección del sistema. En la misma se contempla el MTTD promedio, la cantidad de detecciones y la información sobre el último ataque detectado. La

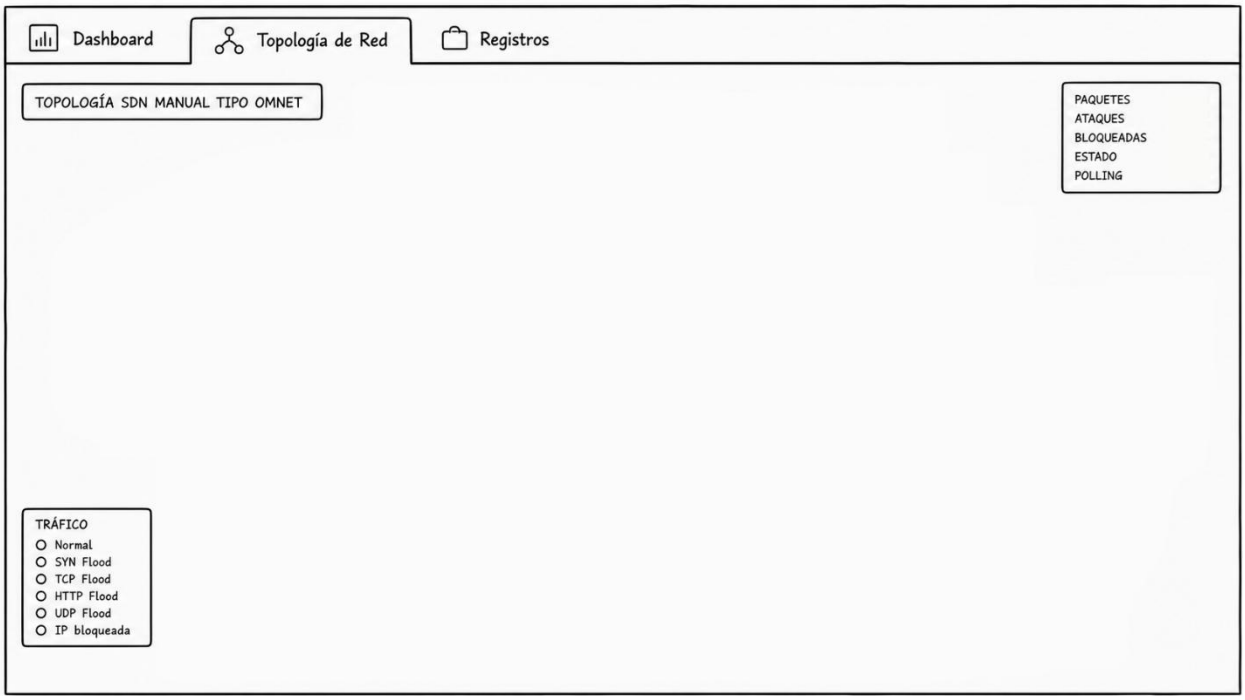
consideración de este indicador facilita la evaluación de la rapidez temporal del sistema para detectar ataques desde su aparición y da una referencia cuantitativa de la eficacia del mecanismo de detección, por lo que el MTTD no solo aporta información, sino que también interviene en la estimación de la calidad de modelo y de la rapidez de respuesta del controlador ante casos de ataque.

Además, integra la sección de predicciones del tráfico en tiempo real, organizada mediante una tabla estructurada. Esta tabla contempla campos como timestamp, dirección IP de origen, dirección IP de destino, protocolo, tipo de tráfico, confianza de la predicción, bytes por segundo, acción aplicada y alcance. También se incluye un filtro para seleccionar los eventos visualizados. Esta estructura permite registrar de forma ordenada cada evento identificado por el sistema, relacionando la clasificación emitida por el modelo con su nivel de confianza y con la acción correspondiente. En consecuencia, el dashboard aporta trazabilidad al proceso de detección y facilita la revisión técnica de los eventos generados durante la operación del sistema.

Posteriormente, como se muestra en la Figura 13 la pestaña Topología de Red está destinada a representar la estructura del entorno SDN monitoreado. En esta sección se contempla una topología SDN manual tipo OMNET, la cual permite ubicar visualmente los elementos que forman parte del escenario de red. Aunque el espacio principal está reservado para la representación gráfica de la topología, también se incluye un panel lateral con información de paquetes, ataques, direcciones bloqueadas, estado y polling. Estos elementos permiten relacionar el comportamiento detectado por el sistema con la condición operativa de la red, fortaleciendo la capacidad de interpretación del operador frente a eventos anómalos.

Figura 13

Diseño del Dashboard del controlador - pestaña Topología de la red



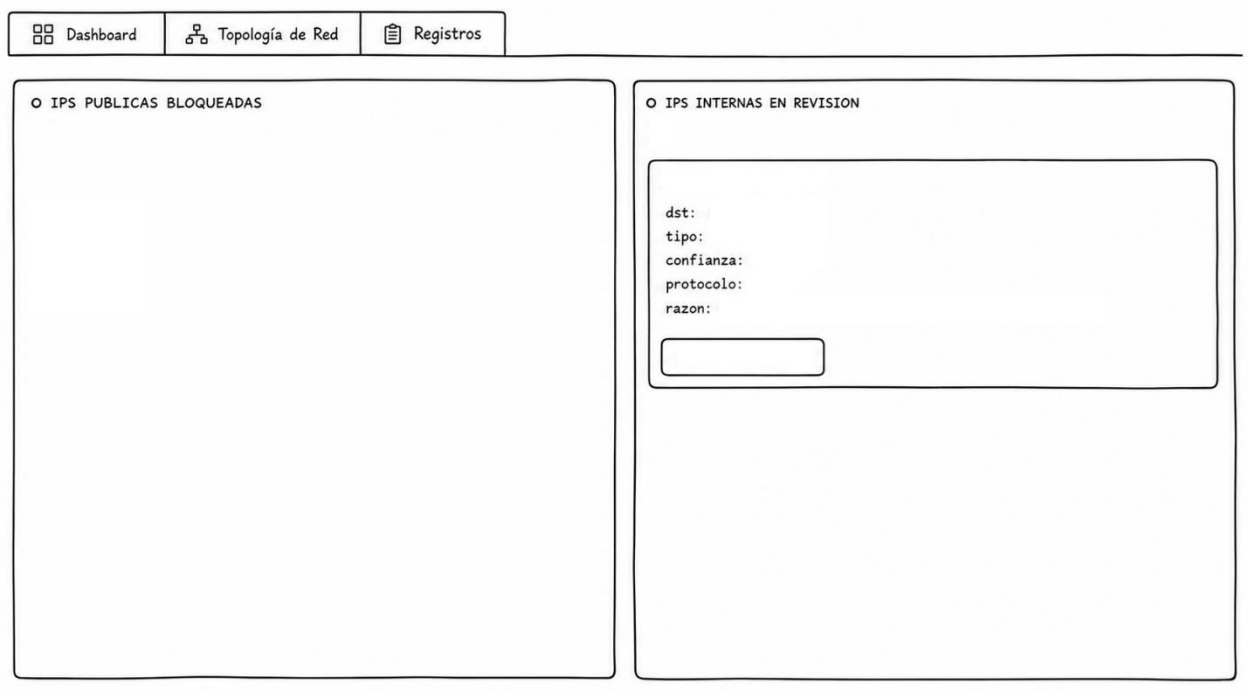
En la misma pestaña se incorpora una leyenda de tráfico que diferencia entre tráfico normal, SYN Flood, TCP Flood, HTTP Flood, UDP Flood e IP bloqueada. Esta clasificación visual es importante porque permite identificar el tipo de evento que puede manifestarse dentro de la topología y distinguir entre tráfico legítimo, tráfico malicioso y acciones de mitigación. De esta manera, la pestaña de topología no solo cumple una función representativa, sino que también actúa como apoyo visual para contextualizar los ataques detectados dentro del entorno SDN.

Finalmente, como se observa en la Figura 14 la pestaña Registros permite dar seguimiento a las acciones defensivas y a las direcciones que requieren revisión dentro del sistema. Esta sección se divide en dos bloques principales: IPs públicas bloqueadas e IPs internas en revisión. El primer bloque funciona como un registro de las direcciones externas que han sido bloqueadas como resultado de las políticas de mitigación aplicadas por el controlador. Esto permite mantener

evidencia de las respuestas ejecutadas automáticamente por el sistema frente a eventos considerados maliciosos.

Figura 14

Diseño del Dashboard del controlador - pestaña Registros



Por otro lado, el bloque de IPs internas en revisión permite analizar eventos que requieren una evaluación adicional antes de aplicar una acción definitiva. Esta área muestra campos de entrada tales como destino, tipo, confianza, protocolo y razón, que son útiles para almacenar criterios respecto la clasificación o revisión de una entrada dada. Esta organización permite mayor trazabilidad al sistema, de manera que existe conocimiento sobre las direcciones que ha bloqueado y sobre qué eventos había en análisis con qué información.

Conjuntamente, el diseño del dashboard incluye cuatro niveles de información de visualización: métricas globales y tráfico en tiempo real, análisis mediante entropía de Rényi y MTTD (Mean Time to Detect), representación de la topología de red y registros de eventos e IPs, que señalan que el sistema de detección basado en Ryu, XGBoost y la entropía de Rényi es

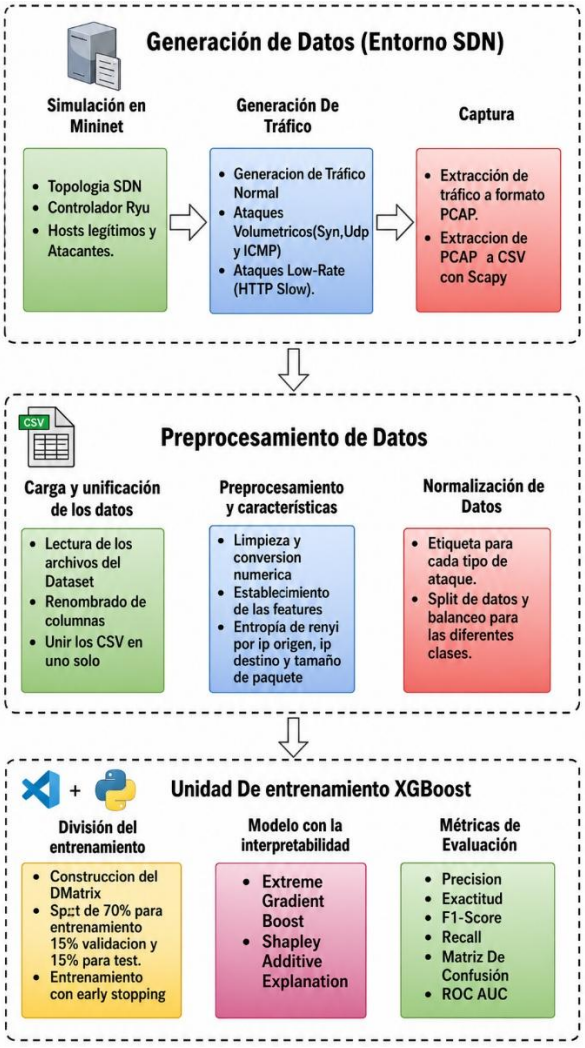
interpretado a partir de una interfaz gráfica que permite la supervisión del tráfico, la validación de predicciones, oposición/identificación de ataques y seguimiento de acciones de mitigación. Por lo tanto, el dashboard es una capa primordial de la arquitectura propuesta, ya que traduce las facetas técnicas del sistema en información visual que resulta útil para la toma de decisiones y la evaluación del mecanismo de defensa frente a ataques DDoS.

3.3.3 Diseño del modelo de detección

3.3.3.1 Diagrama de Bloques del Modelo. El siguiente esquema nos ayuda a visualizar como las diferentes herramientas y tecnologías se integran y colaboraron para poder realizar un análisis eficaz y el proceso de entrenamiento eficiente para el modelo, como se lo puede llegar a interpretar en la siguiente Figura 15.

Figura 15

Diagrama de Bloques del Modelo



Como se pudo observar en la Figura 15 el esquema se encuentra dividido en bloques como: Bloque de generación de datos, bloque de preprocesamiento, bloque de entrenamiento.

Bloque de Generación de Datos (Entorno SDN). Este bloque representa la fase de creación y captura de la fuente de datos fundamental para el desarrollo del sistema de detección. Se simuló un entorno SDN con MiniNet y el controlador Ryu, integrando hosts legítimos y atacantes. Dentro de este entorno se generaron tres perfiles de tráfico: tráfico normal, ataques volumétricos (SYN, UDP e ICMP) y ataques de baja intensidad (HTTP Flood y Slowloris).

Finalmente, el tráfico circulante fue capturado y exportado a formato PCAP, para luego ser transformado a un formato tabular CSV mediante la librería Scapy, constituyendo así el conjunto de datos para el preprocesamiento.

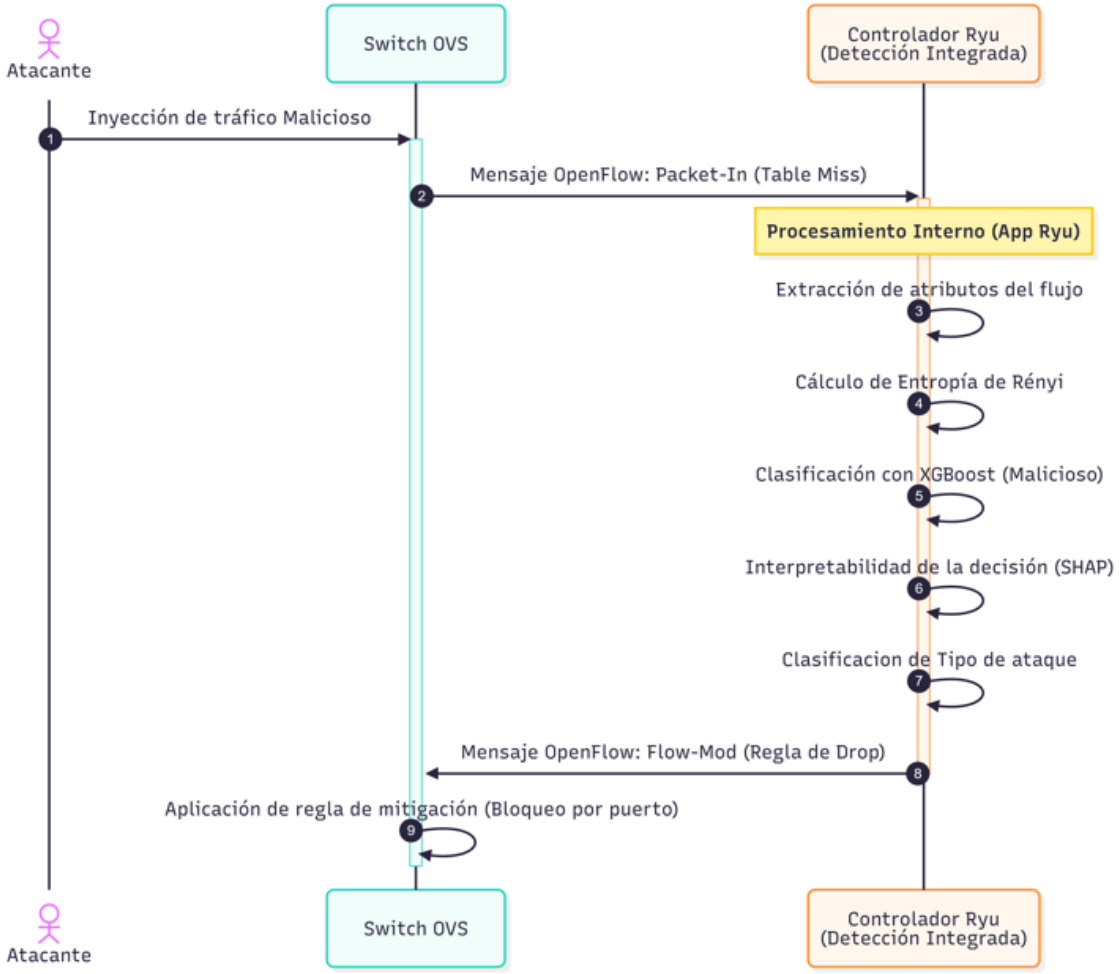
Bloque de Preprocesamiento de Datos. En este bloque se ejecutaron las etapas esenciales para acondicionar y estructurar los datos antes del entrenamiento. En primer lugar, se realizó la carga, lectura y unificación de los múltiples archivos CSV generados, procediendo con el renombrado de columnas para mantener la consistencia. A continuación, en la fase de preprocesado se llevó a cabo una limpieza, así como conversión a numéricas de los datos que conforman las importantes características. En esta etapa se incluye el cálculo de la entropía de Rényi agrupando los valores por las columnas de dirección IP origen, IP destino y tamaño del paquete de modo que se capte la dispersión estadística y se den visibilidad a anomalías sutiles. Por último, se etiquetó cada uno de los tipos de ataque y se aplicaron técnicas de balanceo de las clases que garantizaron la calidad y homogeneización del dataset.

Bloque de Unidad de Entrenamiento XGBoost. Esta fase abarca la estructuración final para el aprendizaje y la evaluación del algoritmo. Inicialmente, se construyó la estructura de datos optimizada DMatrix. Atendiendo a las mejores prácticas de validación de modelos, el conjunto de datos se dividió utilizando una distribución estratificada del 70% para entrenamiento, 15% para validación interna y 15% para la prueba final test. Para el proceso de aprendizaje se empleó el modelo *Extreme Gradient Boost* (XGBoost), usando parada temprana para evitar el sobreajuste. El rendimiento del modelo fue evaluado rigurosamente mediante métricas como precisión, exactitud, F1-score, recall, área bajo la curva (ROC-AUC) y la matriz de confusión. Como complemento indispensable, Se incorporó el análisis SHAP, lo que aporta transparencia al proceso al identificar qué características influyeron mayormente en la clasificación de cada flujo.

3.3.3.2 Diagrama de Secuencia del Sistema de Detección. En el siguiente Esquema se puede visualizar el diagrama de secuencia del sistema de detección para ataques de baja intensidad en SDN y mediante el anterior diagrama de bloques poder tener una detección precisa y el funcionamiento correcto con el siguiente diagrama.

Figura 16

Diagrama de Secuencia del Sistema de Detección



Como se pudo observar en el diagrama de secuencia del sistema de detección de la Figura 16 se muestra la interacción entre los componentes del entorno SDN durante el proceso de identificación y mitigación de ataques. El flujo inicia cuando el atacante genera tráfico malicioso

que es recibido por el switch OVS del plano de datos. Al no encontrar una regla de coincidencia previa, este dispositivo envía mensajes Packet-In al controlador SDN con el fin de solicitar instrucciones sobre el tratamiento del flujo.

Una vez recibido el mensaje, el controlador procesa internamente la información extrayendo las características necesarias para el análisis, eliminando la latencia de enviarlo a un módulo externo. En esta misma etapa, se procesan los atributos relevantes calculando la entropía de Rényi, lo que genera una métrica estadística que describe el nivel de dispersión del tráfico asociado a cada IP origen, IP destino y tamaño del paquete. Toda esta información estructurada es evaluada directamente por el modelo de detección integrado basado en XGBoost, el cual clasifica si el flujo es anómalo o malicioso.

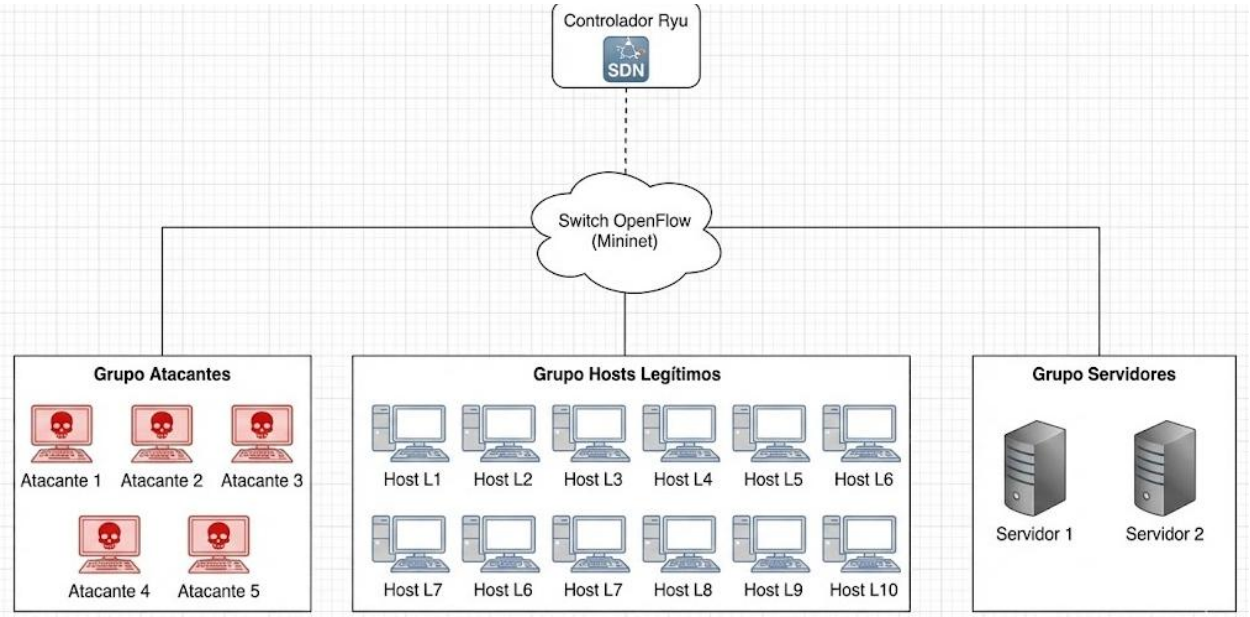
Una vez obtenida la predicción, el sistema ejecuta el submódulo SHAP de manera local para explicar la decisión del modelo, identificando las características con mayor influencia en la clasificación. Con base en esta evaluación interna, el controlador SDN aplica inmediatamente una acción de mitigación, instalando reglas OpenFlow en los switches para bloquear o descartar el tráfico sospechoso por el puerto correspondiente. Por el contrario, si el flujo es clasificado como legítimo, no se aplican restricciones y el tráfico continúa su curso normal dentro de la red.

3.3.4 Diseño del pipeline de datos

3.3.4.1 Origen de datos del Dataset. Para el entrenamiento, validación y evaluación del modelo de detección propuesto, se construyó un conjunto de datos propio. A diferencia de las taxonomías externas, la generación de tráfico a medida en un entorno emulado permitió obtener un control riguroso sobre las tasas de inyección de paquetes, garantizando una representación a las amenazas de denegación de servicio operando sobre una red Definida por Software (SDN).

Figura 17

Arquitectura del banco de pruebas



La recolección de los datos se ejecutó sobre una arquitectura lógica de centro de datos de tres niveles, emulada en la plataforma Mininet bajo la gestión del controlador SDN Ryu. Como se observó en la Figura 17 el entorno integró hosts legítimos que simulaban interacciones y tráfico benigno constante hacia los servidores de la topología, operando junto con una red externa de nodos atacantes. Todo el tráfico circulante fue monitoreado en las interfaces de los conmutadores Open vSwitch (OVS) y capturado en tiempo rCAPeal en formato PCAP (*Packet Capture*) para su posterior análisis.

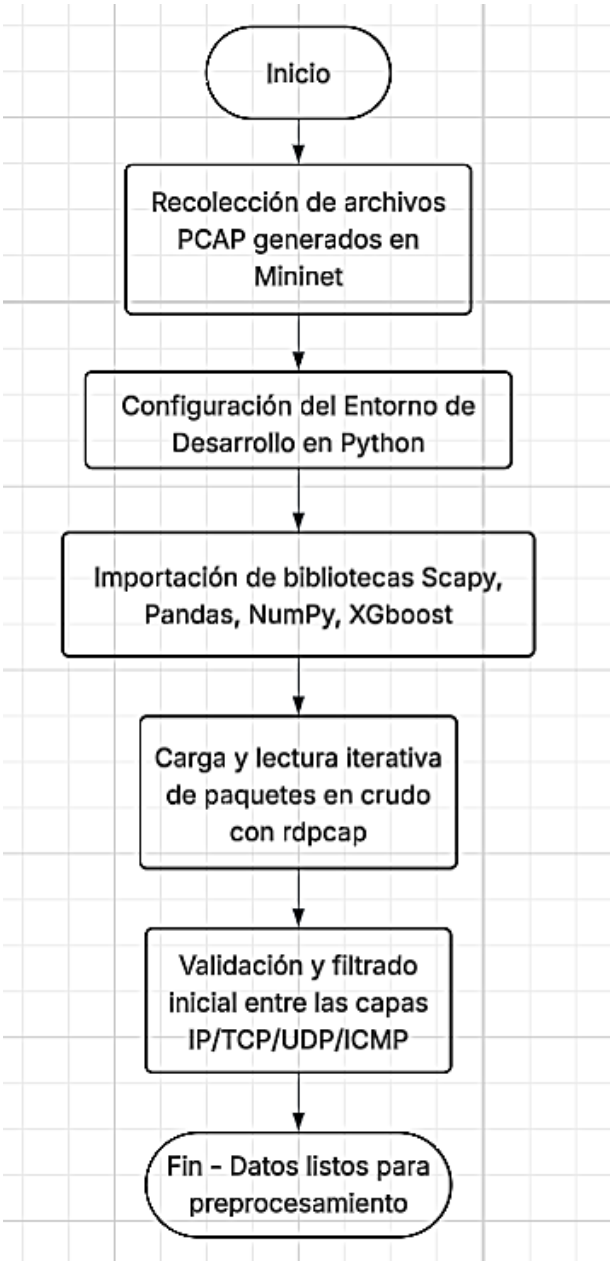
3.3.4.2 Diagramas de Flujo. El diagrama de flujo del sistema diseñado para la detección de ataques ilustra de manera secuencial y lógica las etapas involucradas en el procesamiento del tráfico de red dentro de la arquitectura SDN. Este esquema visual ilustra la transformación de datos desde su captura hasta su clasificación e interpretación. A continuación, se describen los bloques funcionales principales que componen el proceso de detección, cada uno desempeñando un rol crítico en la identificación de anomalías sutiles:

Flujograma de adquisición y carga de datos. El proceso de adquisición inició con la recolección de los archivos de captura de tráfico en crudo formato PCAP que fueron generados previamente en el entorno de simulación Mininet. Estos archivos guardan las seis categorías de tráfico para la investigación: normal, SYN Flood, UDP Flood, ICMP Flood, HTTP Flood y Slowloris. Posteriormente, se realizó la configuración del entorno de desarrollo en Python y la importación de las bibliotecas científicas necesarias para el análisis.

Destaca principalmente el uso de Scapy para la inspección profunda de los paquetes de red, junto con Pandas y NumPy para la estructuración y manipulación de matrices de datos. Cuando el entorno fue preparado, el sistema siguió con la carga y lectura iterativa de cada archivo PCAP que se encontraba ubicado en el directorio local al llamarse la función rdpcap. En esta fase, el algoritmo procedió a validar las capturas y eliminarlas de su memoria, cargando en memoria sólo los paquetes que fuesen válidos y que contuviesen información a nivel de capa de red y de capa de transporte TCP/UDP/ICMP, para cargar los datos en crudo en modo de inicio de ingeniería de características. Esto se lo puede observar claramente en la siguiente Figura 18 en el que se detalla detenidamente el flujograma para la adquisición y carga de los datos.

Figura 18

Flujograma de adquisición y carga de datos



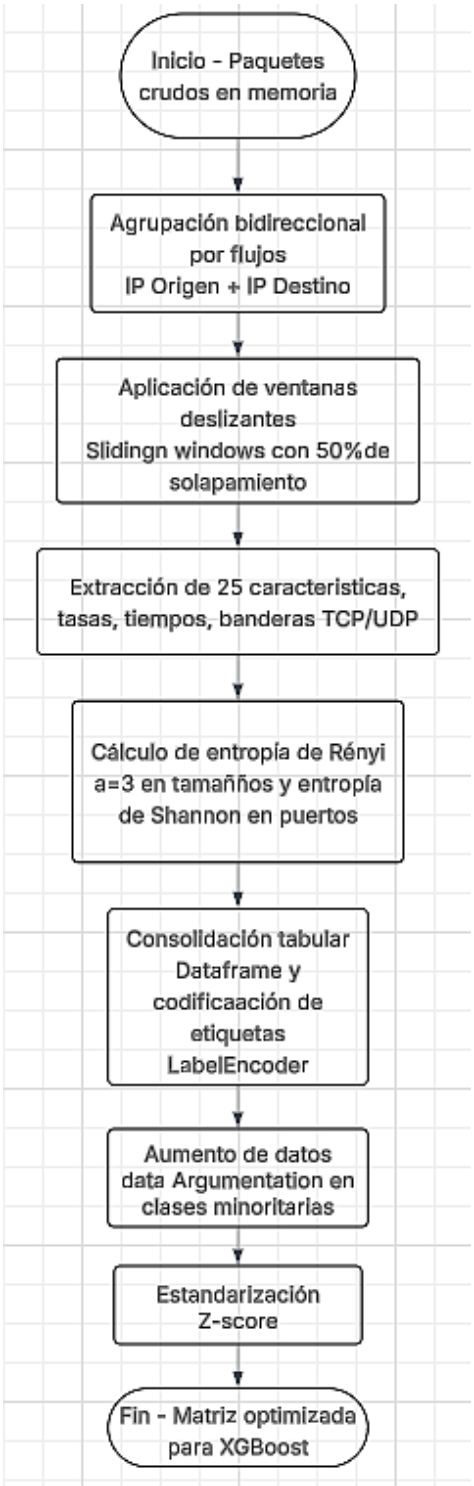
Flujograma de Preprocesamiento de Datos. Durante el preprocesamiento de datos, los paquetes crudos filtrados en la fase anterior fueron sometidos a un riguroso proceso de transformación matemática y estadística. Inicialmente, el tráfico se agrupó en flujos bidireccionales continuos utilizando como identificador único la combinación de la dirección IP de origen y la IP de destino. Para abordar el desafío de la escasez de muestras en ataques sigilosos y capturar su comportamiento temporal, se aplicó un algoritmo de ventanas deslizantes con un

solapamiento programado del 50%, dividiendo los flujos largos en múltiples sub-flujos u observaciones.

Dentro de cada ventana de observación, el sistema ejecutó la ingeniería de características extrayendo 25 métricas clave, tales como tasas de transmisión, varianzas de longitud y conteo de banderas TCP. El núcleo de este bloque fue la aplicación de la teoría de la información: se calculó la Entropía de Rényi con alta sensibilidad sobre la distribución de los tamaños de los paquetes, y la Entropía de Shannon sobre la variabilidad de los puertos de origen y destino. Posteriormente, las características se estructuraron en un formato tabular Datagrama. Para corregir el desbalance inherente entre el tráfico legítimo y los ataques, se implementó una técnica de Aumento de Datos Data Augmentation a las clases minoritarias. Finalmente, las etiquetas fueron codificadas y los datos numéricos pasaron por una estandarización Z-score StandardScaler, resultando en una matriz de características robusta, balanceada y escalada para el algoritmo de clasificación. Todo esto se lo puede evidenciar en la siguiente Figura 19 referenciado al flujograma de preprocesamiento de los respectivos datos.

Figura 19

Flujograma de Preprocesamiento de datos



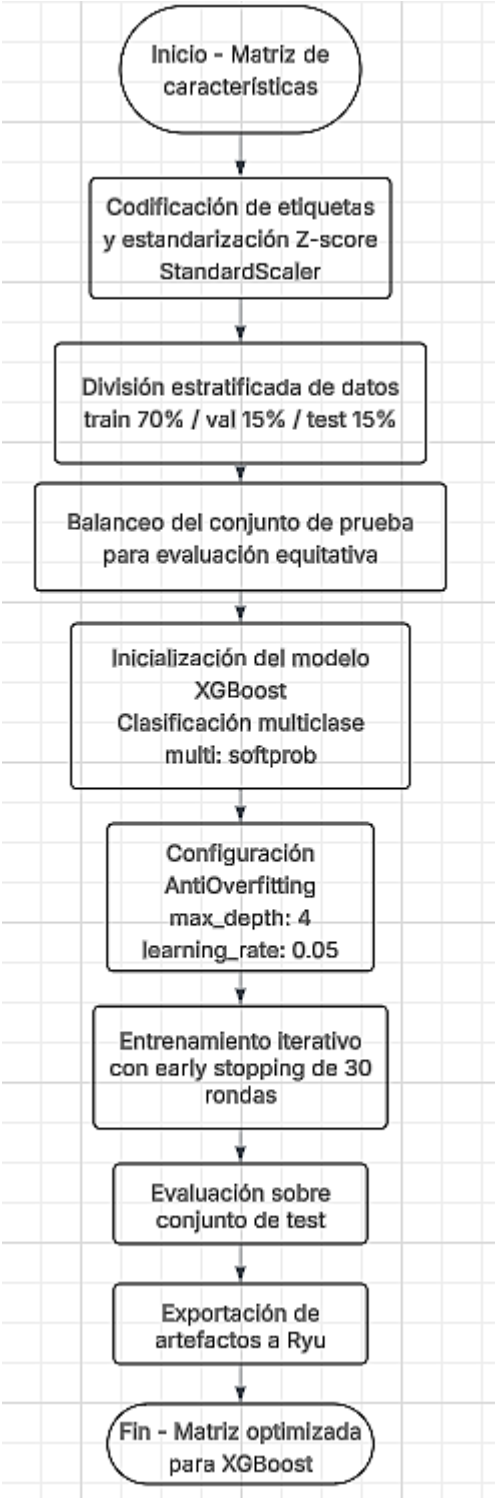
Flujograma del Entrenamiento y Evaluación de Modelo. El proceso formativo comenzó con la codificación de las etiquetas categóricas para las distintas variedades de ataques, así como también para normalizar todas las características numéricas a través de la técnica de estandarización Z-score StandardScaler, que reescala estos atributos para que tiendan a tener una media de cero y una varianza igual a uno. Con la intención de conseguir un aprendizaje estable y evaluar la aptitud en generalización del algoritmo, se estratificó el conjunto de datos en tres segmentos, un set de los datos (70%) se usa para el entrenamiento, el 15% se utiliza para validación interna del aprendizaje, y se reserva un set final de datos (15%) para verificar los resultados después de haber realizado un balanceo estricto.

Una vez estructurados los datos, se inicializó el algoritmo clasificador Extreme Gradient Boosting (XGBoost) configurado para clasificación multiclase. Para mitigar el riesgo de sobreajuste,se aplicó una configuración rigurosa de hiperparámetros, limitando la profundidad máxima de los árboles, ajustando la tasa de aprendizaje y aplicando técnicas de submuestreo. La fase de aprendizaje dinámico incorporó un mecanismo de Parada Temprana (Early Stopping) con una paciencia de 30 iteraciones, el cual monitoreó la pérdida logarítmica multiclase (mlogloss) sobre el conjunto de validación para detener el entrenamiento exactamente en el punto de convergencia óptimo.

Se finalizó el flujo con la evaluación predictiva del modelo sobre el conjunto de prueba aislado. El rendimiento se midió mediante la generación de un informe de clasificación que incluye la Exactitud, Precisión, Recall y F1-Score ponderado, las curvas ROC-AUC para cada clase y una matriz de confusión expresada en porcentajes. El flujo concluyó exportando como artefacto del modelo entrenado empaquetado, dejándolo listo para ser utilizado en tiempo real en la arquitectura de controlador SDN Ryu.

Figura 20

Flujograma de entrenamiento y evaluación del modelo



3.3.4.3 Técnicas de Preprocesamiento y entrenamiento utilizadas. Para el preprocesamiento de los flujos de red aplicadas, se utilizaron técnicas de ciencia de datos en ciberseguridad, como la limpieza de valores outliers, la segmentación en ventanas de tiempo, cálculo de entropía y la normalización de los datos. Las transformaciones de la información dieron lugar a una estructuración eficiente y analítica del tráfico del CSV de entrada para su clasificación mediante el algoritmo XGBoost, optimizando la detección de ataques de baja intensidad.

Estrategia de preprocesamiento de datos. El preprocesamiento de datos se diseñó como una etapa fundamental para garantizar que el tráfico de red capturado pudiera transformarse en una matriz de entrada consistente, discriminativa y apta para el entrenamiento del modelo de detección. En primer lugar, se contempló la limpieza y depuración de registros corruptos, eliminando valores nulos, infinitos y atributos con inconsistencias derivadas de errores de captura o del cálculo de métricas, además de normalizar los nombres de las columnas para preservar la integridad estructural del conjunto de datos. Seguidamente en esta limpieza, se optó también por la aplicación de un filtrado de flujos con intensidades extremas, con el objetivo de focalizar el análisis en patrones sólo sutiles y persistentes, alineados más con lo que serían los comportamientos típicos de los ataques DDoS que poseen baja intensidad.

Seguido a ello el denominado diseño para el preprocesamiento integró un paso de la ingeniería de atributos basado en la segmentación temporal del tráfico y en el cálculo de métricas de entropía, en específico la Entropía de Rényi, para captar los cambios discretos en la distribución de atributos como direcciones IP y puertos, y volcar en cada intervalo de observación una huella cuantificada de los estados de la red, logrando con mayor precisión discriminar entre el tráfico legítimo y los patrones anómalos y, finalmente, la definición de la transformación numérica gracias a la codificación de etiquetas para las clases categóricas y la estandarización Z-score de las

variables continuas, con el fin de homogeneizar las escalas y evitar sesgos en el proceso de aprendizaje. De forma complementaria, se estableció una estrategia de balanceo híbrido con SMOTE y Random UnderSampler, aplicada exclusivamente sobre el conjunto de entrenamiento, para corregir el desbalance entre clases y mejorar la capacidad del modelo para aprender de manera robusta los comportamientos minoritarios asociados a eventos maliciosos.

Estrategia de entrenamiento del modelo. La estrategia de entrenamiento del modelo se diseñó con el propósito de construir un clasificador multiclase capaz de diferenciar tráfico legítimo y distintas variantes de ataques DDoS de baja intensidad a partir de características estadísticas y entrópicas extraídas del tráfico de red. Con este propósito, se realizó una partición del conjunto de datos de forma estratificada en lo que respecta a subconjuntos de entrenamiento, de validación y de entrenamiento, de forma que se preservase la proporción de clases originales y al mismo tiempo se mantuviese una evaluación del desempeño del modelo homogénea. A partir de esta, se utilizó el algoritmo de clasificación XGBoost, siendo este capaz de aprender cómo manejar las relaciones no lineales, además de ser robusto frente a datasets heterogéneos y ser de excelente calidad en problemas de detección con múltiple número de variables de entrada. Igualmente, se consideró el uso de ponderación por clase y balanceo del conjunto de entrenamiento con la finalidad de reducir el sesgo hacia las clases de mayor tamaño, así como mejorar la sensibilidad del sistema ante patrones anómalos y menos representativos.

La estrategia de entrenamiento empleada se complementó con un ajuste de hiperparámetros cuyo objetivo era controlar la complejidad del modelo y evitar el sobreajuste mediante criterios de máxima profundidad, tasa de aprendizaje, submuestreo y regularización. Además, se consideró un mecanismo de validación temprana para detener el entrenamiento cuando el desempeño dejara de mejorar sobre el conjunto de validación, favoreciendo así la generalización del clasificador.

Finalmente, se estableció la exportación de los artefactos del modelo entrenado incluyendo el clasificador, el escalador, el codificador de etiquetas y la lista de características con el propósito de integrarlos posteriormente al controlador Ryu y reproducir en tiempo real el mismo esquema de inferencia definido durante la fase offline de entrenamiento.

3.3.5 Diseño del sistema de ataques

3.3.5.1 Taxonomía de ataques. La topología del entorno de ataques se diseñó para reproducir dentro de la red SDN emulada, distintos comportamientos maliciosos representativos de ataques DoS/DDoS, tanto de carácter volumétrico como de baja intensidad. Para ello, se definió una taxonomía operativa de ataques que permitió organizar los vectores utilizados en el laboratorio según su mecanismo de afectación, distinguiendo entre patrones de saturación bruta, como UDP Flood, ICMP Flood y SYN Flood, y patrones de degradación progresiva, como HTTP Flood, y SlowHTTP. Esta clasificación permitió estructurar el entorno de ataque no solo desde el punto de vista funcional, sino también desde una perspectiva metodológica, al asociar cada tipo de tráfico malicioso con el comportamiento que debía ser observado por el controlador y posteriormente aprendido por el modelo de detección. Por tal razón al encontrarse una gran variedad de ataques a través de la Figura 21 se explica cada uno de forma respectiva y como dentro de cada aspecto puede llegar a afectar dicha situación.

Figura 21

Taxonomía de ataques DDoS clasificado por Volumétricos y por Baja intensidad



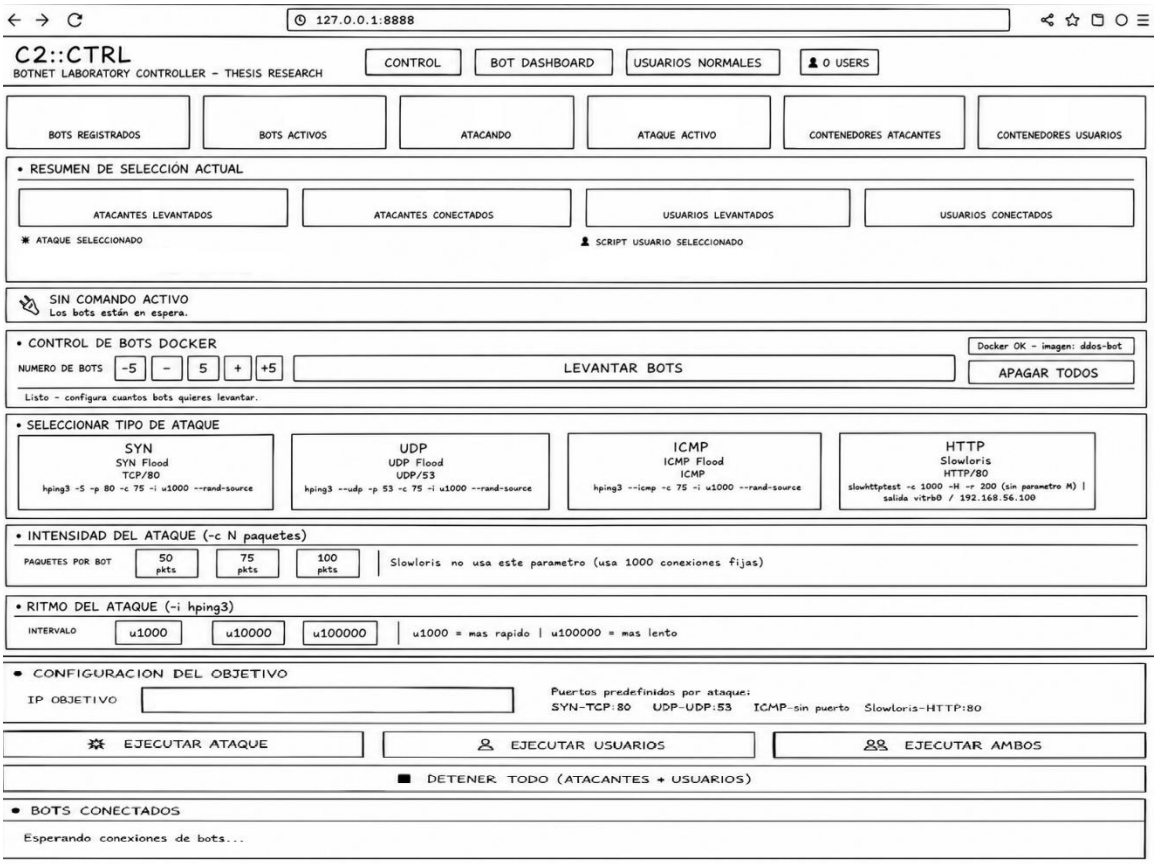
3.3.5.2 Diseño del Dashboard de administración de atacantes. El dashboard de administración de atacantes se diseñó como una interfaz de control operativo destinada a gestionar, de forma centralizada, la generación de tráfico malicioso y tráfico legítimo dentro del entorno experimental. Su propósito principal es proporcionar un mecanismo estructurado para orquestar la activación de nodos atacantes, definir el tipo de ataque a ejecutar, configurar la intensidad del tráfico, establecer el objetivo de la prueba y controlar la participación de usuarios normales. A diferencia de un panel meramente informativo, este dashboard cumple una función activa dentro del sistema, ya que permite ejecutar acciones directamente sobre los contenedores Docker utilizados en el laboratorio, facilitando la construcción de escenarios controlados para la generación del dataset y la validación del modelo de detección.

Como se observa en la Figura 22 la primera pestaña del dashboard corresponde al módulo Control del sistema C2::CTRL – Botnet Laboratory Controller. Esta sección constituye el panel

principal de administración de atacantes, debido a que concentra los indicadores generales del entorno experimental y las opciones necesarias para ejecutar ataques simulados. En la parte superior se presentan indicadores como bots registrados, bots activos, estado de ataque, ataque activo, contenedores atacantes y contenedores de usuarios. Estos elementos permiten conocer rápidamente la condición operativa del laboratorio antes de iniciar una prueba, verificando si existen nodos disponibles, si se encuentran conectados y si hay algún comando activo en ejecución.

Figura 22

Diseño del Dashboard de administración de ataques - primera pestaña



Dentro de esta misma pestaña se incorpora un resumen de selección actual, donde se visualiza la cantidad de atacantes levantados, atacantes conectados, usuarios levantados y usuarios conectados. Esta información permite validar que la infraestructura experimental se encuentre

correctamente preparada antes de generar tráfico. Asimismo, se muestra un estado de comando activo, en el cual se indica si los bots se encuentran en espera o si existe una instrucción en ejecución. Este componente resulta importante porque permite diferenciar entre un entorno inactivo y un escenario en el que los nodos atacantes están ejecutando una acción específica.

El dashboard también incluye una sección denominada Control de Bots Docker, desde la cual se puede definir el número de bots atacantes que serán levantados en el entorno. Para ello, la interfaz presenta controles de incremento y decremento, así como opciones rápidas para aumentar la cantidad de nodos. Se añaden los botones levantar bots y apaga todos, que permiten gestionar los contenedores atacantes de forma que no es necesario intervenir en la infraestructura Docker. Esto otorga flexibilidad al entorno de pruebas, dado que se puede aumentar el nivel de distribución del ataque en función de las necesidades en cada experimento.

Otro elemento importante de la pestaña Control es la mención del tipo de ataque. En esta parte se presentan distintas clases de tráfico malicioso, entre las cuales se encuentran SYN Flood, UDP Flood, ICMP Flood y HTTP Slowloris. Cada uno de éstas representa un comportamiento de ataque distinto, por lo que permite crear diferentes escenarios de ataque a partir de los cuáles es posible evaluar el comportamiento del sistema de detección de ataques. De este modo, el dashboard permite la simulación de ataques volumétricos y de utilidad de recursos, y de este modo se puede comprobar como el controlador SDN y el modelo de clasificación se comportan según variaciones en el tráfico de ataque.

La interfaz también permite configurar la potencia del ataque a partir de la selección de paquetes por bot y la velocidad del ataque, que se define a partir del intervalo de tiempo utilizado por las herramientas hping3. Estos parámetros son fundamentales para controlar la agresividad del tráfico generado, ya que permiten variar la cantidad de paquetes enviados y la velocidad de

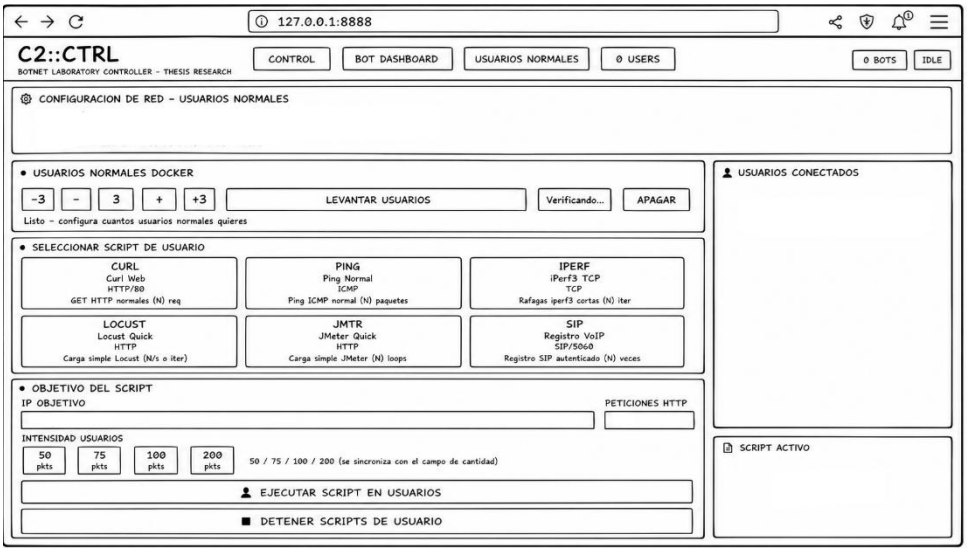
emisión. De esta forma, el entorno experimental no se limita a ejecutar ataques fijos, sino que permite ajustar la intensidad de las pruebas para observar el comportamiento del sistema ante escenarios de baja, media o mayor carga.

En la parte inferior de la pestaña se encuentra la configuración del objetivo, donde se define la dirección IP objetivo del ataque. También se define una lista de puertos predefinidos de acuerdo con el tipo de ataque escogido, como por ejemplo SYN/TCP, UDP, ICMP o HTTP para Slowloris, por lo que se convierten en conexiones que simplifican el instanciamiento del escenario experimental permitiendo asociar cada ataque con su protocolo y puerto correspondiente. La interfaz también incorpora botones para ejecutar ataque, ejecutar usuarios, ejecutar todos, parar todo, lo que permite ejecutar las acciones del laboratorio de forma centralizada o detenerlas. La sección de bots conectados permite verificar si los nodos atacantes se han enlazado correctamente al controlador de comandos.

Como se muestra en la Figura 23 la segunda pestaña corresponde al módulo de Usuarios Normales, diseñado para administrar la generación de tráfico legítimo dentro del entorno experimental. Esta sección complementa la simulación de ataques, ya que permite introducir tráfico benigno en la red y de esta manera construir escenarios más realistas para la evaluación del sistema de detección. Su función principal es controlar usuarios desplegados en contenedores Docker y ejecutar scripts que representan comportamientos normales de red.

Figura 23

Diseño del Dashboard de administración de ataques - segunda pestaña



En esta pestaña se presenta una sección de Usuarios Normales Docker, donde se puede definir la cantidad de usuarios legítimos que serán levantados. Al igual que en el módulo de atacantes, la interfaz dispone de controles para aumentar o disminuir el número de usuarios, así como botones para levantar usuarios y apagar los contenedores correspondientes. En el costado derecho se incluye un panel de usuarios conectados, el cual permite verificar si los nodos benignos se encuentran activos y disponibles para ejecutar los scripts configurados.

La sección de selección de script de usuario permite escoger diferentes tipos de tráfico legítimo. Entre las opciones visibles se incluyen CURL, orientado a peticiones HTTP normales; PING, utilizado para generar tráfico ICMP normal; IPERF, destinado a pruebas de tráfico TCP; LOCUST y JMETER, empleados para simular carga HTTP y SIP, asociado a registros de VoIP. La presencia de estas opciones permite generar tráfico benigno diversificado, lo cual es importante para evitar que el dataset esté compuesto únicamente por tráfico malicioso. De esta forma, el sistema puede ser evaluado en condiciones más cercanas a una red real, donde coexisten comunicaciones normales y eventos anómalos.

Además, el módulo de usuarios normales incorpora la configuración del objetivo del script, donde se define la IP objetivo y, en el caso correspondiente, la cantidad de peticiones HTTP. La pestaña de la interfaz donde se permite manejar la intensidad de usuarios también está presente, y permite utilizar valores como 50, 75, 100 y 200 paquetes, que son configurables. Esto permite definir la cantidad de tráfico legítimo que se genera haciendo posible así la creación de situaciones donde el tráfico normal tenga diferentes niveles de actividad. Esta funcionalidad es importante porque permite saber si el modelo de detección puede poner en correcto lugar el tráfico benigno intenso del tráfico maligno.

Por otro lado, la pestaña contiene botones que permiten ejecutar script en usuarios y detener scripts de usuario; además de un panel de script activo. Con ello se permite saber qué actividad legítima se está ejecutando en un momento dado y detenerla en un momento dado. En resumen, permite complementar un poco la funcionalidad de la pestaña del dashboard, permite equilibrar el entorno experimental a partir de crear tráfico normal de manera controlada, por lo que también sirve para mejorar el sistema de recogida de datos del proceso de validación del sistema.

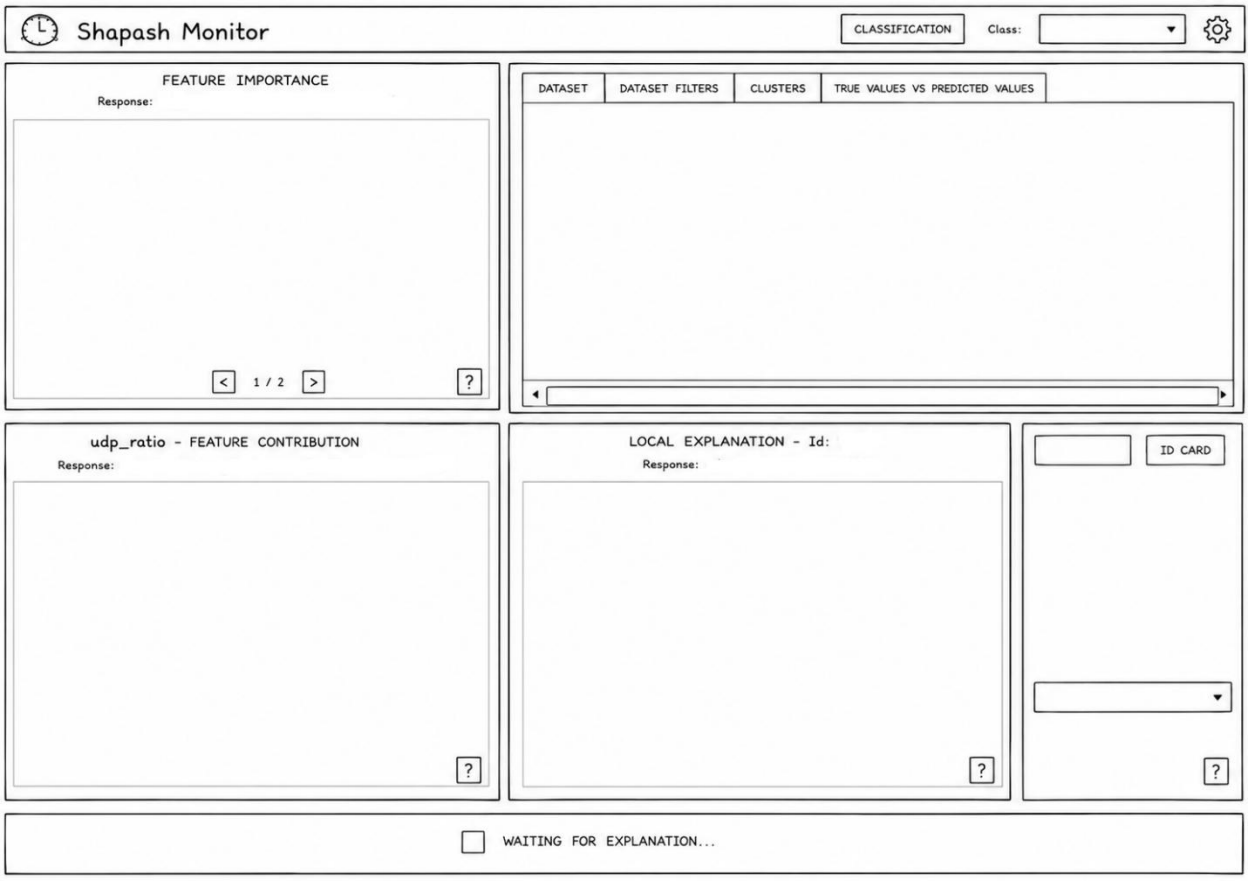
Dicho de otra forma, el dashboard de administración de atacantes integra dos funciones básicas; la orquestación de bots malignos y la generación de tráfico legítimo con usuarios normales. La pestaña Control permite orquestar ataques DDoS simulados configurando la cantidad de bots, el tipo de ataque, la intensidad, el ritmo y el objetivo. La pestaña Usuarios Normales permite levantar usuarios benignos y ejecutar scripts de tráfico legítimo a partir de herramientas como CURL, PING, IPERF, LOCUST, JMETER y SIP. Esta organización convierte al dashboard en una capa de comando y control experimental, capaz de administrar de forma centralizada los escenarios necesarios para la construcción del dataset, la evaluación del controlador SDN y la validación del modelo de detección propuesto.

3.3.6 Diseño del sistema del modelo interpretable SHAPASH

El dashboard del modelo interpretable fue diseñado como una interfaz de supervisión destinada a facilitar el análisis de las predicciones generadas por el modelo XGBoost. Su estructura se basa en el dashboard proporcionado por SHAPASH y mantiene una distribución modular, de manera que el usuario pueda consultar tanto el comportamiento general del modelo como la explicación de una predicción específica. La interfaz se divide en cuatro áreas principales como se observa en la Figura 24.

Figura 24

Diseño del Dashboard del sistema del modelo interpretable SHAPASH



3.3.6.1 Cabecera del sistema. Incluye el nombre del módulo denominado SHAPASH Monitor, el tipo de tarea ejecutada, que corresponde a clasificación multiclase y un selector de clase. Este selector permite analizar de manera independiente las categorías normales, HTTP Flood, UDP Flood, SYN Flood e ICMP Flood.

3.3.6.2 Importancia global de características. Esta sección presenta un gráfico de barras horizontales ordenado según la contribución media absoluta de cada variable. Busca identificar las características que más influyen en las decisiones del modelo.

3.3.6.3 Exploración del conjunto de datos. Incluye una tabla interactiva con las observaciones del explicador. Muestra los valores de entrada, la clase real y la predicha. Permite seleccionar un registro para analizar su explicación local.

3.3.6.4 Contribución de características. Muestra la relación existente entre el valor de una variable y su aporte en la predicción; lo que nos permitirá ver si una característica aumenta o disminuye la probabilidad de pertenecer a una clase dada.

3.3.6.5 Explicación local. Muestra la contribución individual de las variables para un registro seleccionado. Las contribuciones positivas favorecen la clase predicha, mientras que las negativas reducen su probabilidad. En el panel lateral se incorporan controles para seleccionar el índice del registro, el umbral de visualización y la cantidad de características mostradas.

3.3.7 Tabla resumen de librerías pensadas a implementar

Con el propósito de sustentar técnicamente la arquitectura del sistema propuesto, se definió un conjunto de librerías y dependencias de software orientadas a cubrir las funciones de emulación de red, control SDN, procesamiento de tráfico, aprendizaje automático, almacenamiento intermedio y visualización. La Tabla 21 resume las principales librerías empleadas, su descripción general, el módulo donde se integran y la función específica que cumplen dentro del sistema.

Tabla 21

Librerías empleadas

Librería	Descripción	Módulo o componente donde se utiliza	Función dentro del sistema
Mininet	Framework de emulación de redes SDN.	Generación del dataset y entorno de red.	Desplegar la topología, hosts legítimos, atacantes y enlaces de prueba.
Ryu	Framework para controladores SDN basados en OpenFlow.	Controlador principal del sistema.	Gestionar eventos de red, reglas OpenFlow y acciones de mitigación.
Scapy	Librería para lectura y manipulación de paquetes.	Pipeline offline de datos.	Leer archivos PCAP y extraer tráfico IP/TCP/UDP/ICMP.
NumPy / Pandas	Librerías de cómputo numérico y análisis tabular.	Pipeline offline y dashboard.	Procesar métricas, estructurar ventanas y consolidar el dataset.
Scikit-learn	Biblioteca de preprocesamiento y evaluación de modelos.	Pipeline offline.	Codificar etiquetas, escalar variables, dividir datos y calcular métricas.
XGBoost	Librería de clasificación supervisada basada en gradient boosting.	Entrenamiento offline e inferencia en Ryu.	Entrenar y ejecutar el modelo multiclase de detección de ataques.
imbalanced-learn (SMOTE, RandomUnderSampler)	Biblioteca para balanceo de clases.	Pipeline offline.	Corregir el desbalance del conjunto de entrenamiento.

Librería	Descripción	Módulo o componente donde se utiliza	Función dentro del sistema
Joblib	Librería de serialización de objetos.	Pipeline offline y controlador Ryu.	Guardar y cargar el modelo, escalador y codificador de etiquetas.
Redis	Base de datos en memoria y canal de intercambio de eventos.	Controlador y dashboard de visualización.	Compartir métricas, alertas, logs e IPs bloqueadas en tiempo real.
Streamlit / Plotly	Framework web y librería de visualización interactiva.	Dashboard de visualización del controlador.	Mostrar topología, métricas de red, entropía de Rényi y ataques detectados.
Flask / Docker SDK	Framework web y cliente de administración de contenedores.	Dashboard de administración de atacantes.	Gestionar bots Docker, emitir comandos y configurar campañas de ataque.
Bibliotecas estándar de Python (json, os, time, threading, socket, pathlib, collections, etc.)	Módulos nativos de soporte general.	Todos los componentes del sistema.	Manejar archivos, tiempos, hilos, estructuras de datos, red y utilidades auxiliares.

Además de las librerías descritas en la tabla, el sistema emplea herramientas externas como iperf, hping3, tcpdump, slowhttptest, wget, ping y Docker Engine para la generación de tráfico legítimo y malicioso, la captura en formato PCAP y la orquestación del entorno experimental de atacantes. Estas herramientas forman parte del soporte operativo del laboratorio, pero no se consideran librerías Python dentro del desarrollo del sistema.

3.4 Implementación

La implementación del sistema de detección de ataques se desarrolló a través de un proceso secuencial que inició con la generación de tráfico sintético y su captura en formato PCAP dentro de un entorno emulado. A partir de estos datos, se construyó el pipeline de preprocesamiento mediante un script en Python, el cual ejecutó la extracción de características utilizando ventanas deslizantes, el cálculo de entropía de Rényi para los tamaños de paquete y entropía de Shannon para los puertos, además del balanceo de clases y la estandarización de variables. Dicha primera fase finalizó tras el proceso de entrenamiento, evaluación y exportación de los artefactos del modelo clasificador XGBoost, tal y como se encontraba optimizado para su puesta en marcha.

A continuación, se pasó al despliegue de la infraestructura de Red Definida por Software, configurando la topología lógica y los switches Open vSwitch (OVS). Uno de los hitos de esta etapa fue la integración de dos controladores Ryu que debían funcionar al unísono para conseguir balanceo de carga cuya finalidad era asegurar la alta disponibilidad y la tolerancia a fallos del plano de control. Finalmente, se pasó a construir el escenario de pruebas mediante la integración del modelo XGBoost previamente exportado como una aplicación nativa de estos controladores, lo que permite a través de ellos hacer un continuo scraping de flujos de red, o realizar detección y mitigación de amenazas en tiempo real.

3.4.1 Implementación de la topología

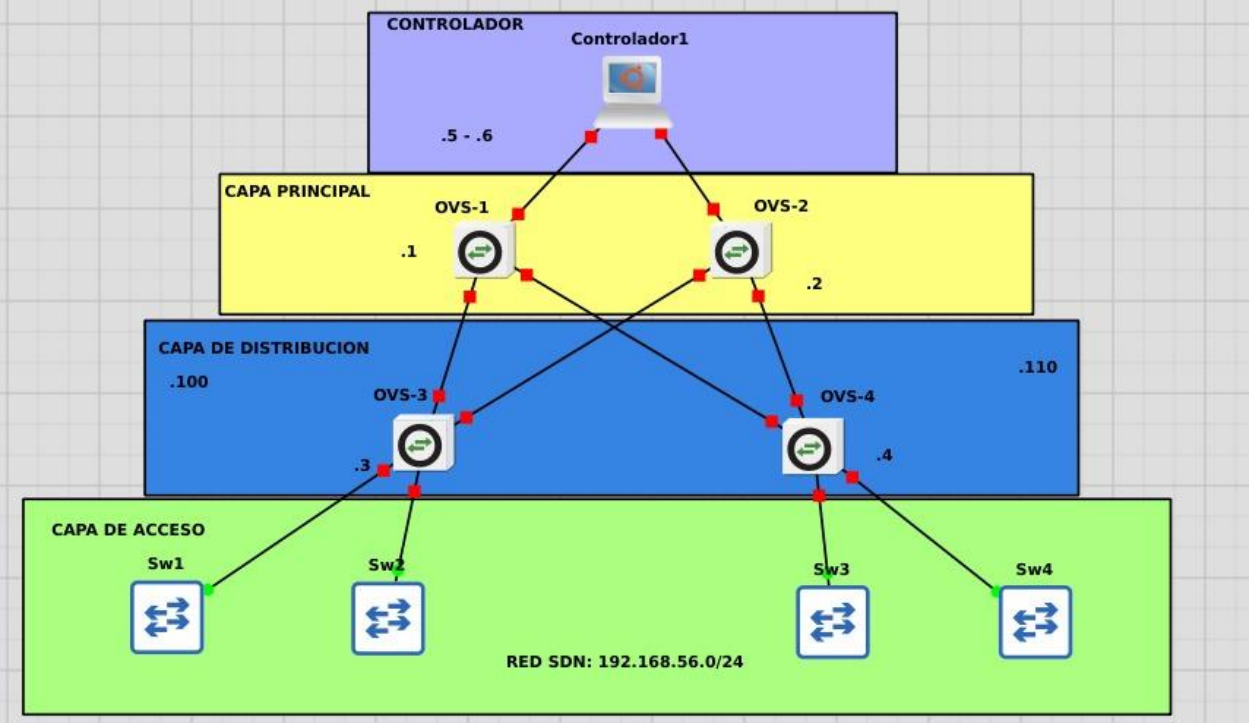
La aplicación de la topología de red emuló el entorno de SDN previamente planteado en la fase de diseño, para darle una infraestructura emulada con tráfico legítimo y tráfico malicioso dentro del mismo escenario de experimentación. Para ello, se utilizó Mininet como plataforma de despliegue, integrando un controlador remoto Ryu, un conmutador Open vSwitch y un conjunto de nodos organizados según su función dentro del laboratorio. Esta disposición permitió construir

una base operativa estable para la generación del dataset y para la posterior integración del mecanismo de detección en tiempo real.

3.4.1.1 Despliegue de dispositivos y enlaces de la red SDN. El despliegue de la red SDN inicia al asociar los conmutadores OVS y conectar sus puertos en GNS3. Este procedimiento busca interconectar los switches con el controlador para lograr una gestión centralizada de la red. La Figura 25 muestra los enlaces para la topología de red propuesta.

Figura 25

Establecimiento de los enlaces de la red SDN



Tras definir la topología de la red SDN, se deben identificar los puertos conectados al controlador para asignarles una dirección IPv4 según la Tabla 22. Esto busca enlazar los puertos de los conmutadores SDN y comunicarlos con el controlador a través de OpenFlow según la Tabla 19.

Tabla 22

Comandos que permiten la asociación de los switches SDN al controlador

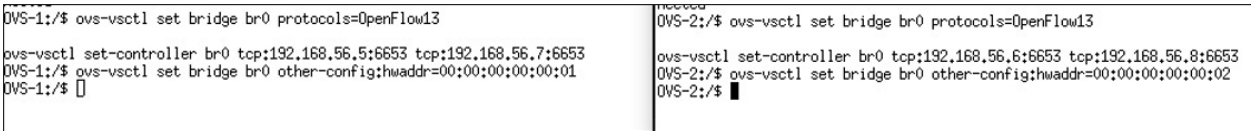
Sentencia	Descripción
ovs-vsctl set-controller br0 tcp:<ip>:<puerto>	Define la dirección IP y el puerto TCP (6633 o 6653) del controlador SDN al que se conectará el switch.
ovs-vsctl set bridge br0 other-config:datapath-id=<DPID>config:datapath id=000000000000000#	Asigna un Identificador de Ruta de Datos Datapath ID único en formato hexadecimal. Este identificador es crucial para que el controlador reconozca cada dispositivo dentro de la topología SDN.
ovs-vsctl set bridge br0 protocols=OpenFlow13	Restringe y fuerza al switch a utilizar la versión 1.3 del protocolo OpenFlow, asegurando la compatibilidad con características necesarias para la recolección de estadísticas multiruta.

Nota: Adaptado de (Open vSwitch, 2024)

En la Figura 26 se muestran los comandos de Open Vswitch que configuran la asociación de conmutadores SDN al controlador. Estas configuraciones se replicarán en los otros switches, ajustando solo el hardware según el número de switch.

Figura 26

Configuraciones de los enlaces OVS al controlador



Por otro lado, en cuanto a la parte de código se puede mencionar que el despliegue inicial consistió en crear programáticamente la topología experimental mediante Mininet, definiendo el controlador remoto, el switch principal y los enlaces hacia hosts legítimos y nodos atacantes. Esta

implementación permitió concentrar el tráfico en un punto común de observación y simplificó la interacción entre el plano de datos y el plano de control. La elección de una topología centralizada sobre un switch principal respondió a la necesidad de capturar, redirigir y analizar el tráfico en un entorno controlado, pero suficientemente representativo para validar el comportamiento del sistema de detección.

3.4.1.2 Implementación de la topología para el conjunto de datos con Mininet

Figura 27

Implementación de la estructura base del entorno SDN

```python
def crear_topologia():
    net = Mininet(controller=RemoteController, switch=OVSKernelSwitch, link=TCLink)

    info('*** Agregando Controlador Ryu\n')
    # Definimos el controlador pero esperamos a start() para conectarlo
    c0 = net.addController('c0', controller=RemoteController, ip=CONTROLLER_IP, port=6633)

    info('*** Agregando Switch s1\n')
    s1 = net.addSwitch('s1')

    info('*** Agregando Hosts Legítimos (h1-h5)\n')
    legitimos = []
    for i in range(1, 6):
        # Usamos h11, h12 para evitar confusiones internas de mininet
        h = net.addHost(f'h1{i}', ip=f'10.0.0.{i}')
        net.addLink(h, s1, bw=10)
        legitimos.append(h)

    info('*** Agregando Atacantes (a1-a3)\n')
    atacantes = []
    for i in range(1, 4):
        a = net.addHost(f'a{i}', ip=f'10.0.0.{20+i}')
        net.addLink(a, s1, bw=10)
        atacantes.append(a)

    info('*** Iniciando la red\n')
    net.start()
    return net, legitimos, atacantes
```

Como se pudo observar en la Figura 27 el fragmento de código implementa la estructura base del entorno SDN y define la conexión de todos los nodos al switch virtual, estableciendo así el laboratorio sobre el cual se desplegaron los escenarios de tráfico y las rutinas de detección.

3.4.1.3 Configuración del direccionamiento IPv4 y conectividad del conjunto de datos.

Una vez desplegados los dispositivos, se procedió a la asignación del direccionamiento IPv4 dentro del segmento experimental y a la validación de conectividad entre nodos. En esta fase, la numeración lógica de los hosts permitió diferenciar claramente entre nodos legítimos y nodos atacantes, favoreciendo la trazabilidad del tráfico durante la captura y el análisis. Como parte del proceso se contempló una etapa de calentamiento y verificación inicial mediante pingAll, con el objetivo de establecer flujos previos, confirmar la conectividad con el controlador Ryu y evitar inestabilidades al inicio del escenario. Como se lo puede interpretar en la siguiente Figura 28.

Figura 28

Configuración del direccionamiento IPv4 y conectividad del conjunto de datos

```
CONTROLLER_IP = '127.0.0.1'
for i in range(1, 6):
    # Usamos h11, h12 para evitar confusiones internas de mininet
    h = net.addHost(f'h1{i}', ip=f'10.0.0.{i}')
for i in range(1, 4):
    a = net.addHost(f'a{i}', ip=f'10.0.0.{20+i}')
try:
    # 2. FASE DE CALENTAMIENTO (CRUCIAL PARA QUE RYU NO SE CIERRE)
    info('*** Esperando 5 segundos para conexión con Ryu...\n')
    time.sleep(5)

    info('*** Ejecutando PingAll para establecer flujos (Evita colapso de Ryu)...\n')
    net.pingAll()
    time.sleep(2)
```

Este procedimiento aseguró que la red estuviera operativa antes de iniciar la captura de tráfico y la ejecución de ataques, estableciendo una condición de partida consistente para todos los escenarios experimentales.

3.4.2 Implementación de la arquitectura del controlador y monitoreo

La implementación de la arquitectura del controlador y monitoreo se centró en integrar la lógica de análisis, clasificación y respuesta dentro del plano de control SDN, utilizando Ryu como núcleo de procesamiento. Simultáneamente, se llevaron a cabo dos la construcción de dos

interfaces complementarias: un dashboard de visualización orientado hacia la observabilidad del propio sistema, y otro dashboard, el de administración de atacantes, orientado hacia la orquestación de bots o de campañas experimentales. Esta arquitectura hizo posible unificar control, visibilidad y administración de operaciones en el contexto del entorno de pruebas.

3.4.2.1 Implementación de la aplicación del controlador Ryu. La ejecución de la arquitectura del controlador y monitorización se enfocó en integrar la lógica de análisis, clasificación y respuesta dentro del plano de control SDN, siendo Ryu el núcleo del procesamiento. En conjunto con ello, se desarrollaron dos interfaces complementarias en la integración del sistema; un dashboard de visualización, orientado a la visualización gráfica del sistema, y un dashboard de gestión de atacantes, orientado a la orquestación de bots y para campañas experimentales. Esta arquitectura permitió unificar control, visibilidad y administración operativa dentro del entorno de pruebas.

Figura 29

Implementación de la aplicación del controlador Ryu

```python
class DDoSController(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        logging.getLogger().setLevel(logging.ERROR)
        self.logger.setLevel(logging.INFO)

        self.ml = MLInference(MODEL_DIR)
        self.feature_extractor = TrainingAlignedFeatureExtractor(self.ml.renyi_alpha)
        self.window_buffer = PerDstWindowBuffer(self.ml.window_size)
        self.renyi_l2 = RenyiLayer2(alpha=self.ml.renyi_alpha)
        self.redis_logger = RedisLogger()

        self.mac_to_port = {}
        self._blocked = {}
        self._protected = set()
        self._last_attack_ts = 0.0
```

Este bloque define la base operativa del controlador e integra los módulos que posteriormente permiten ejecutar la inferencia y la mitigación en línea.

3.4.2.2 Gestión de flujos y eventos Packet-In. La gestión de flujos se implementó a partir del registro del switch en OpenFlow 1.3 y de la instalación de una regla por defecto que reenvía el tráfico al controlador, como se lo está llegando a visualizar en la siguiente Figura 30. Esta decisión permitió que los primeros paquetes de cada flujo fueran inspeccionados por la lógica de detección antes de determinar una acción definitiva sobre ellos. De esta manera, el controlador pudo recibir eventos Packet-In, extraer atributos de red y alimentar el proceso de clasificación a partir de observaciones reales del tráfico circulante.

Figura 30

Gestión de flujos

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    dp = ev.msg.datapath
    ofp = dp.ofproto
    parser = dp.ofproto_parser

    self._add_flow(
        dp, 0, parser.OFPMatch(),
        [parser.OFPActionOutput(ofp.OFPP_CONTROLLER, ofp.OFPCML_NO_BUFFER)]
    )
    self.logger.info(f"[Switch] dp_id={dp.id} conectado")
```

A nivel interno, el controlador mantuvo un buffer por destino para conservar un historial reciente de paquetes asociados a una misma IP de destino, permitiendo la construcción de ventanas de observación consistentes con el pipeline de entrenamiento.

Figura 31

Gestión de eventos Packet-In

```python
class PerDstWindowBuffer:
    def __init__(self, window_size: int):
        self.window_size = window_size
        self.buffers = defaultdict(list)

    def add_packet(self, ip_src, ip_dst, size, ts, proto, sport, dport, tcp_flags=0):
        self.buffers[ip_dst].append({
            "src": ip_src,
            "dst": ip_dst,
            "size": float(size),
            "ts": float(ts),
            "proto": proto,
            "sport": int(sport),
            "dport": int(dport),
            "tcp_flags": int(tcp_flags),
        })
```

3.4.2.3 Implementación del Dashboard de visualización del controlador. El dashboard se creó con Streamlit y Redis para eventos y métricas. Este panel permite consultar logs recientes, topología, estadísticas, IPs bloqueadas y métricas del sistema. El dashboard visualiza el estado del controlador en elementos entendibles para el operador.

Figura 32

Implementación del Dashboard de visualización del controlador

```
@st.cache_resource
def get_redis():
    try:
        r = redis.Redis(host=REDIS_HOST, port=REDIS_PORT, db=REDIS_DB,
                        decode_responses=True, socket_timeout=3)
        r.ping(); return r
    except Exception as e:
        st.error(f"Redis no disponible: {e}"); return None
```

Para alimentar dinámicamente la visualización de la topología y los eventos recientes, se implementó además un servidor HTTP auxiliar que expone los datos extraídos desde Redis en formato JSON. Esto permitió desacoplar la captura de eventos del ciclo de actualización del frontend y mejorar la fluidez del monitoreo.

Figura 33

Implementación del servidor HTTP auxiliar

```
class _TopoHandler(BaseHTTPRequestHandler):
    _redis_conn = None

    def do_GET(self):
        if self.path in ('/topo', '/topo/'):
            try:
                r = _TopoHandler._redis_conn
                if r is None:
                    raise RuntimeError("no redis")

                # Last 30 log entries
                raw = r.lrange(R_LOGS, 0, 29) or []
                evts = []
                for item in raw:
                    try:
                        e = json.loads(item)
                        evts.append({
                            "src":     e.get("src_ip",""),
                            "dst":     e.get("dst_ip",""),
                            "lbl":     e.get("label","benign"),
                            "disp":    e.get("display_label","Normal"),
                            "atk":     bool(e.get("is_attack",False)),
                            "conf":    float(e.get("confidence",0)),
                            "ts":      e.get("timestamp",""),
                            "in_port": int(e.get("in_port",0)),
                            "dp_id":   int(e.get("dp_id",1)),
                        })
                    except: pass
```

La representación topológica del dashboard incluyó nodos del controlador, switches OVS, servidores, usuarios y atacantes, de modo que la visualización no solo reflejara métricas abstractas, sino también la estructura real del entorno experimental. De forma visual se puede llegar a observar en la siguientes Figuras 34, 35, 36 y 37, respectivamente.

Figura 34

Dashboard implementado del controlador - primera pestaña

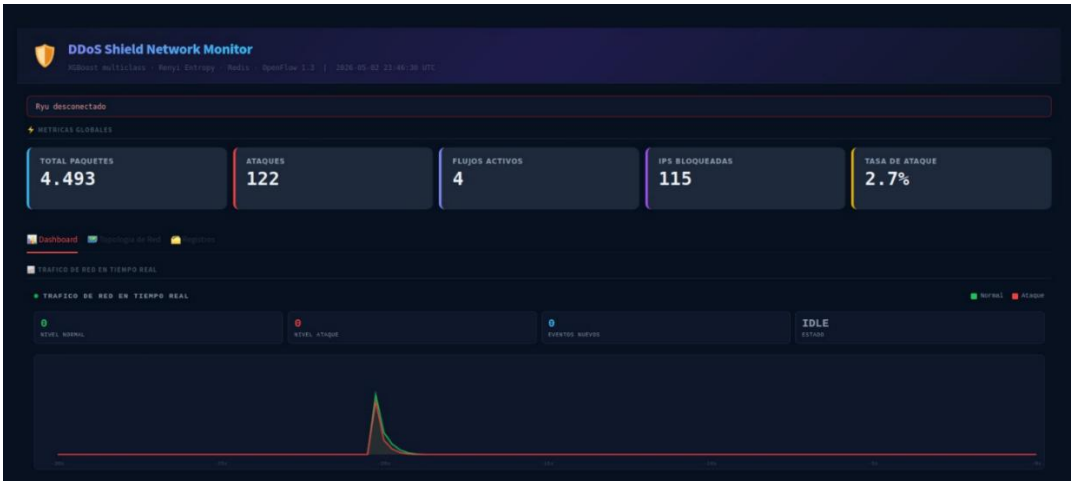


Figura 35

Dashboard implementado del controlador - pestaña Indicadores técnicos

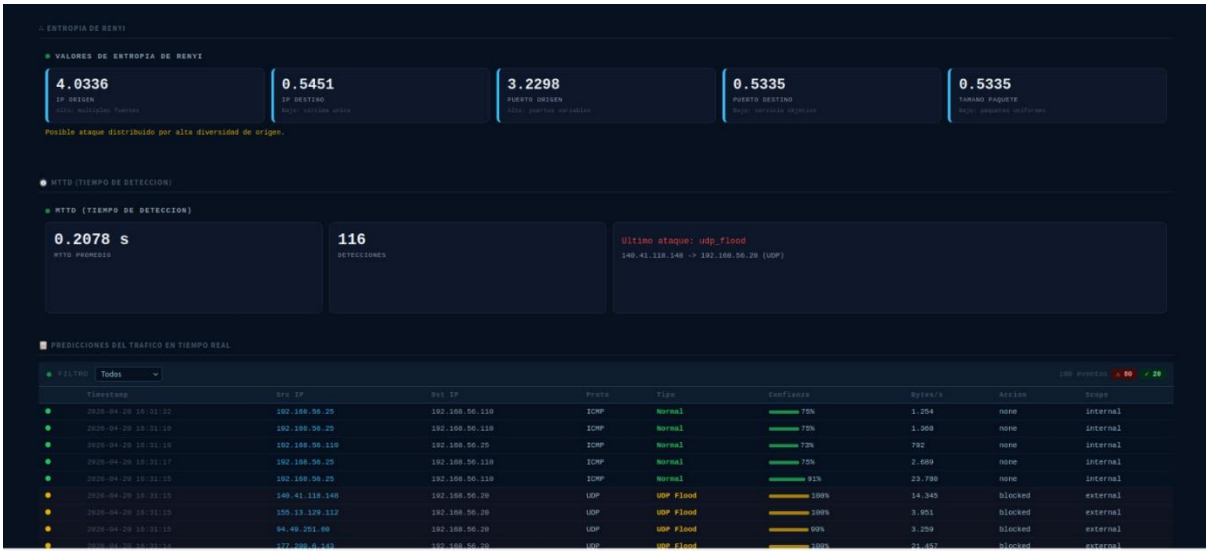


Figura 36

Dashboard implementado del controlador - pestaña sobre la topología de la red

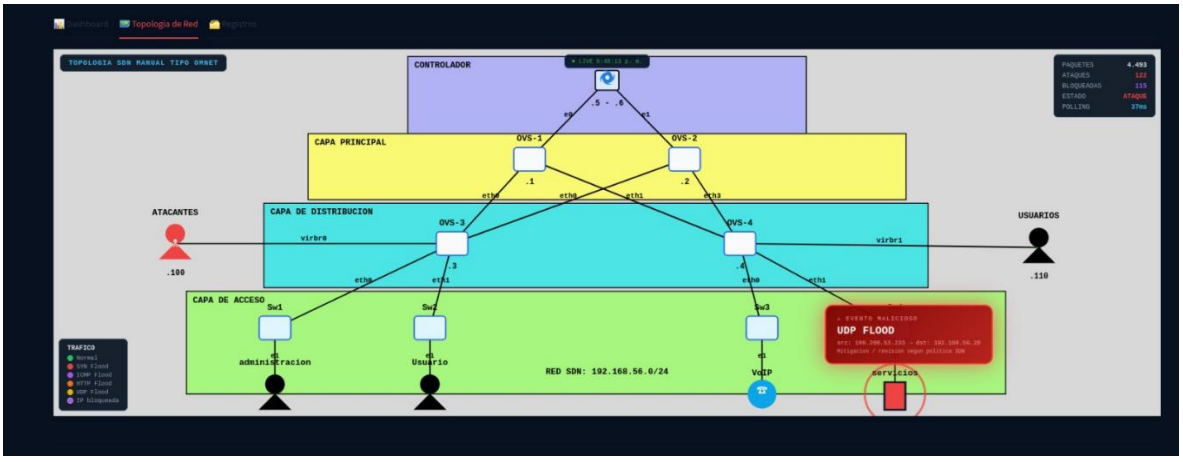
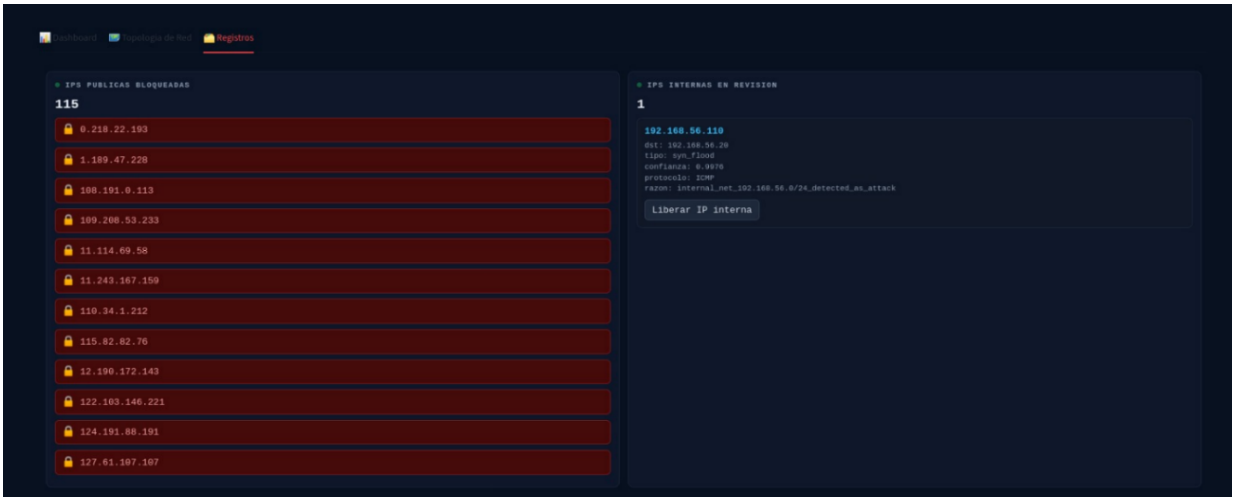


Figura 37

Dashboard del controlador - pestaña Registros



3.4.2.4 Implementación de la aplicación del controlador Ryu. El dashboard de administración de atacantes se implementó mediante Flask y Docker SDK, permitiendo controlar desde una interfaz centralizada el despliegue de bots de laboratorio y la ejecución de campañas de ataque. Esta aplicación mantiene un estado global del sistema, registra bots activos, conserva historial de comandos y permite emitir órdenes coordinadas hacia los nodos maliciosos. Su diseño lo convierte en un componente de control activo, no únicamente en una interfaz de visualización.

Figura 38

Dashboard implementado de administración de ataques - primera pestaña

```
state = {
    "command":    None,
    "issued_at":  None,
    "bots":       {},       # bot_id -> {ip, last_seen, status, executions}
    "history":    [],
    "containers": [],       # lista de container IDs gestionados por el C2
    "bot_count":  0,        # cuantos bots se pidio levantar
}

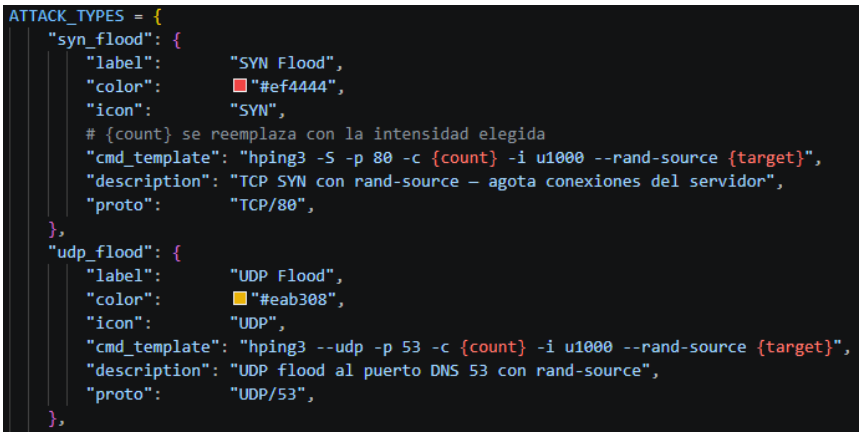
# Nombre de la imagen Docker del bot
BOT_IMAGE      = "ddos-bot"
BOT_NAME_PREFIX = "c2-bot"
# IP del C2 accesible desde los contenedores (--network host)
C2_SELF_URL    = os.environ.get("C2_URL", "http://192.168.56.100:8888")
lock = threading.Lock()

# Intensidades disponibles — controlan el parametro -c de hping3
INTENSIDADES = [50, 75, 100, 200]
INTENSIDAD_DEFAULT = 75
```

De igual forma, se definieron plantillas de ataque que permiten seleccionar el tipo de tráfico malicioso, la intensidad y el objetivo de la campaña, facilitando la generación controlada de SYN Flood, UDP Flood, ICMP Flood y otros patrones dentro del laboratorio.

Figura 39

Dashboard implementado de administración de ataques - segunda pestaña



```
ATTACK_TYPES = {
    "syn_flood": {
        "label":       "SYN Flood",
        "color":       "#ef4444",
        "icon":        "SYN",
        # {count} se reemplaza con la intensidad elegida
        "cmd_template": "hping3 -S -p 80 -c {count} -i u1000 --rand-source {target}",
        "description": "TCP SYN con rand-source — agota conexiones del servidor",
        "proto":       "TCP/80",
    },
    "udp_flood": {
        "label":       "UDP Flood",
        "color":       "#eab308",
        "icon":        "UDP",
        "cmd_template": "hping3 --udp -p 53 -c {count} -i u1000 --rand-source {target}",
        "description": "UDP flood al puerto DNS 53 con rand-source",
        "proto":       "UDP/53",
    },
```

La interfaz también contempla control de bots Docker, selección de intensidad y activación o detención de campañas, lo que mejora la repetibilidad y trazabilidad de los escenarios experimentales.

3.4.3 Implementación del modelo de detección

La interfaz también contempla control de bots Docker, selección de intensidad y activación o detención de campañas, lo que mejora la repetibilidad y trazabilidad de los escenarios experimentales.

3.4.3.1 Integración del cálculo de entropía en el sistema. La implementación del modelo de detección se orientó a integrar la lógica analítica del sistema dentro del entorno SDN operativo. Para ello, se reutilizaron los artefactos generados durante el entrenamiento offline y se adaptó la extracción de características al flujo de paquetes procesados por el controlador. De este modo, la inferencia, la evaluación de confianza y la aplicación de contramedidas quedaron integradas en una misma cadena funcional dentro del controlador Ryu.

Figura 40

Implementación del cálculo de entropía al sistema en el modelo de detección

```
features = {
    "duration":              duration,
    "iat_mean":              float(np.mean(iat)),
    "iat_std":               float(np.std(iat)),
    "iat_max":               float(np.max(iat)),
    "iat_min":               float(np.min(iat)),

    "packet_count":          int(len(sizes)),
    "total_bytes":           float(np.sum(sizes)),
    "pkt_len_mean":          float(np.mean(sizes)),
    "pkt_len_std":           float(np.std(sizes)),
    "pkt_len_max":           float(np.max(sizes)),
    "pkt_len_min":           float(np.min(sizes)),
    "pkt_len_var":           float(np.var(sizes)),

    "pkt_rate":              len(sizes) / max(duration, 0.001),
    "byte_rate":             float(np.sum(sizes)) / max(duration, 0.001),

    "tcp_ratio":             pc.get("TCP",  0) / total,
    "udp_ratio":             pc.get("UDP",  0) / total,
    "icmp_ratio":            pc.get("ICMP", 0) / total,

    "syn_count":             syn_count,
    "ack_count":             ack_count,
    "fin_count":             fin_count,
    "rst_count":             rst_count,

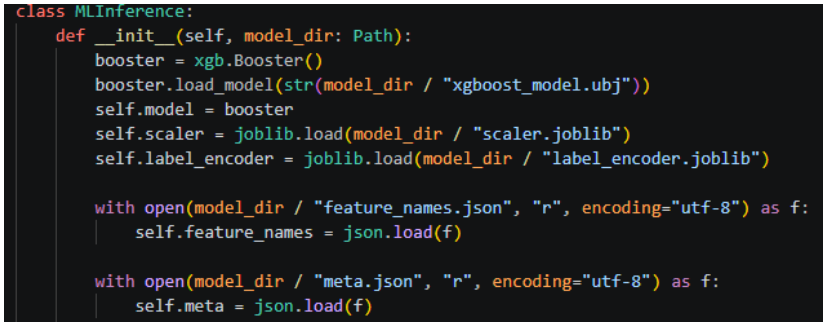
    "unique_ips_src":        unique_ips_src,
    "unique_ports_src":      len(set(ports_src.tolist())),
    "unique_ports_dst":      len(set(ports_dst.tolist())),
    "src_diversity_ratio":   unique_ips_src / max(len(ips_src), 1),
    "ip_src_concentration":  top_src_count / max(len(ips_src), 1),

    "renyi_ip_src":          renyi_entropy(ips_src.tolist(), self.renyi_alpha),
    "renyi_ip_dst":          renyi_entropy([dst_int] * len(sizes), self.renyi_alpha),
    "renyi_pkt_len":         renyi_entropy(sizes.tolist(), self.renyi_alpha),
```

3.4.3.2 Implementación del modelo XGBoost. El modelo XGBoost fue implementado en el controlador gracias a que se cargó el clasificador exportado, el escalador y el codificador de etiquetas, lo que permite que la inferencia en tiempo real pueda utilizar exactamente el mismo espacio de características definido en la fase de entreno evitando así que el entorno offline y el entorno operativo sean inconsistentes entre sí.

Figura 41

Implementación del modelo XGBoost en el modelo de detección



```
class MLInference:
    def __init__(self, model_dir: Path):
        booster = xgb.Booster()
        booster.load_model(str(model_dir / "xgboost_model.ubj"))
        self.model = booster
        self.scaler = joblib.load(model_dir / "scaler.joblib")
        self.label_encoder = joblib.load(model_dir / "label_encoder.joblib")

        with open(model_dir / "feature_names.json", "r", encoding="utf-8") as f:
            self.feature_names = json.load(f)

        with open(model_dir / "meta.json", "r", encoding="utf-8") as f:
            self.meta = json.load(f)
```

3.4.3.3 Implementación del flujo de inferencia en tiempo real. Una vez cargado el modelo, el flujo de inferencia en tiempo real se desarrolló a partir de la extracción de características por ventana y su transformación al formato esperado por XGBoost. Luego del escalado, el sistema generó una predicción multiclase y obtuvo tanto la etiqueta de salida como la confianza asociada a cada observación. Esta información fue posteriormente utilizada para el registro del evento, la visualización en dashboard y la toma de decisiones de mitigación.

Figura 42

Implementación del flujo de inferencia en tiempo real en el modelo de detección

```python
def predict(self, features: dict):
    try:
        vec = np.array(
            [features.get(fn, 0.0) for fn in self.feature_names],
            dtype=np.float64
        ).reshape(1, -1)

        vec = np.nan_to_num(vec, nan=0.0, posinf=0.0, neginf=0.0)
        vec_scaled = self.scaler.transform(vec)
        dmatrix = xgb.DMatrix(vec_scaled, feature_names=self.feature_names)
        prob = self.model.predict(dmatrix)[0]

        idx = int(np.argmax(prob))
        label = self.label_encoder.inverse_transform([idx])[0]
        conf = float(prob[idx])
```

3.4.3.4 Implementación de las acciones de mitigación. Finalmente, la mitigación fue implementada mediante una tabla de políticas que asocia cada tipo de ataque con una acción específica, prioridad e intervalo de validez. Gracias a esta estructura, el sistema pudo responder de forma diferenciada según la naturaleza de la amenaza detectada, aplicando bloqueo, limitación de tasa o ninguna acción cuando se trataba de tráfico benigno. Esta decisión permitió vincular directamente la clasificación del modelo con una respuesta operativa sobre la red.

Figura 43

Implementación de las acciones de mitigación en el modelo de detección

```python
MITIGATION = {
    "http_flood": {"action": "rate_limit", "priority": 200, "idle_timeout": 60},
    "icmp_flood": {"action": "block",      "priority": 300, "idle_timeout": 30},
    "syn_flood":  {"action": "block",      "priority": 300, "idle_timeout": 30},
    "udp_flood":  {"action": "block",      "priority": 300, "idle_timeout": 30},
    "benign":     {"action": "none",       "priority": 0,   "idle_timeout": 0},
    "normal":     {"action": "none",       "priority": 0,   "idle_timeout": 0},
}
```

3.4.4 Implementación del pipeline de datos

La implementación del pipeline de datos constituyó la fase encargada de transformar el tráfico crudo generado en el entorno SDN en una representación estructurada y apta para el entrenamiento del modelo de detección. El proceso empieza con la obtención de capturas de tráfico en formato PCAP obtenidas desde el escenario de evaluación, seguido del enriquecimiento

estadístico y matemático de los flujos y acababa con el entrenamiento y la exportación del clasificador XGBoost a integrar con el controlador Ryu, logrando así que el pipeline actuara como un puente entre la generación del tráfico en la red emulada y la operación posterior del sistema de detección en tiempo real.

3.4.4.1 Implementación del origen y adquisición de datos. El proceso de adquisición de datos se llevó a cabo realizando la ejecución de escenarios controlados de tráfico en el entorno Mininet. De esta manera se levantó la topología de experimentación, el tráfico legítimo de fondo fue generado, la captura en formato PCAP fue iniciada y, por último, se inyectaron distintos tipos de ataques con el objetivo de crear un conjunto de datos representativo. Dicha organización facilitó la posibilidad de disponer de los archivos independientes de tráfico normal y de cada categoría de ataque, logrando así la trazabilidad de los registros y su posterior transformación en un dataset estructurado.

Como se pudo observar en la Figura 28 presentada en el punto 3.4.1.3 el fragmento se mostró la base estructural del laboratorio, sobre la cual se construyeron las sesiones de captura y generación de tráfico. La topología se compone de un controlador remoto, un switch principal, cinco hosts legítimos y tres nodos atacantes, lo que permitió reproducir un entorno SDN controlado, pero suficientemente flexible para generar múltiples patrones de tráfico.

Figura 44

Implementación del origen y adquisición de datos

```python
def capturar_trafico(net, nombre_archivo):
    s1 = net.switches[0]
    info(f'*** INICIANDO CAPTURA: {nombre_archivo}\n')
    s1.cmd(f'timeout {TIEMPO_ESCENARIO} tcpdump -i s1-eth1 -w {DIR_SALIDA}/{nombre_archivo} &')

def ejecutar_ataque(tipo, atacantes, victima):
    info(f'*** EJECUTANDO ATAQUE: {tipo} ---> {victima.IP()}\n')

    if tipo == "udp_flood":
        for a in atacantes:
            a.cmd(f'timeout {TIEMPO_ESCENARIO} hping3 --udp -p 80 -i u10000 {victima.IP()} &')

    elif tipo == "syn_flood":
        for a in atacantes:
            a.cmd(f'timeout {TIEMPO_ESCENARIO} hping3 -S -p 80 -i u10000 {victima.IP()} &')

    elif tipo == "icmp_flood":
        for a in atacantes:
            a.cmd(f'timeout {TIEMPO_ESCENARIO} hping3 --icmp -i u10000 {victima.IP()} &')

    elif tipo == "slowloris":
        atacante_principal = atacantes[0]
        atacante_principal.cmd(f'slowhttptest -c 500 -H -g -o slowloris_stats -i 10 -r 200 -t GET -u http://{victima.IP()}:80 -l {TIEMPO_ESCENARIO} &')
```

Como se pudo observar en la Figura 44 en este segundo bloque se observa el mecanismo práctico de adquisición: primero se habilita la captura del tráfico sobre la interfaz del switch y luego se ejecuta el ataque correspondiente desde los nodos maliciosos. Así, cada archivo PCAP conserva una relación directa con un escenario definido, lo que facilita la separación posterior entre tráfico benigno y tráfico malicioso.

Figura 45

Carga del Dataset y categorías Generadas

```
print("\nDatasets cargados:")
print(list(dfs.keys()))

Cargado: http_flood.csv | Clase: http_flood | Shape: (154184, 27)
Cargado: udp_flood.csv | Clase: udp_flood | Shape: (174421, 27)
Cargado: syn_flood.csv | Clase: syn_flood | Shape: (111843, 27)
Cargado: icmp_flood.csv | Clase: icmp_flood | Shape: (97354, 27)
Cargado: normal.csv | Clase: normal | Shape: (136620, 27)

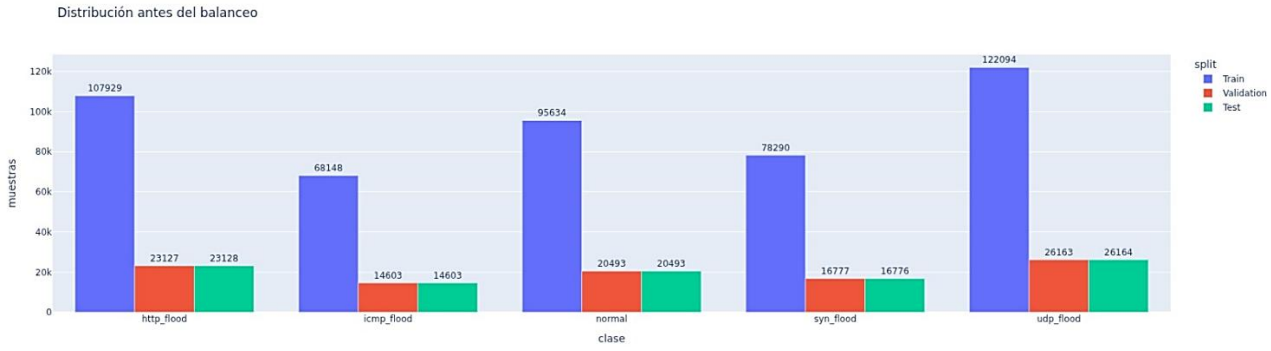
Datasets cargados:
['http_flood', 'udp_flood', 'syn_flood', 'icmp_flood', 'normal']
```

En el siguiente paso, se realizó una exploración exhaustiva del conjunto de datos unificado para obtener una visión general de su dimensionalidad y estructura tras la simulación en la topología Mininet como se lo pudo observar en la Figura 45. Este procedimiento fundamental permitió corroborar la correcta integración de los flujos de red, revelando un volumen total de 640,496 muestras distribuidas en 30 características (*features*) numéricas, las cuales incluyen parámetros volumétricos, conteo de banderas TCP y las métricas avanzadas de Entropía de Rényi.

Como parte de esta fase, se ejecutó el conteo de ocurrencias para cada valor único en la variable objetivo, evidenciando que el tráfico se divide en seis categorías distintas con una marcada disparidad en su frecuencia. Específicamente, la clase del tráfico legítimo encabezó el conjunto de muestras, alcanzando 252,323 (39.4% del total), seguida por ataques volumétricos tales como UDP Flood con 160,521 muestras (25.1%), SYN Flood 101,193 (15.8%) e ICMP Flood con 91,257 muestras (14.2%). En otro lado, los ataques de aplicación y de baja intensidad dejaron una huella mucho más reducida; HTTP Flood alcanzó 32,620 muestras (5.1%) y el ataque Slowloris apenas alcanzó 2,582 muestras, que aun así alcanzaron un 0.4% crítico de la total. Este análisis de distribuciones evidenció empíricamente el desbalance de clases que caracteriza profundamente a los ataques DDoS silenciosos. Conocer esta distribución descompensada fue per se esencial, ya que justificó técnicamente la necesidad de poner en práctica en las fases posteriores algoritmos de sobre muestreo sintético combinados con submuestreo aleatorio para garantizar que el modelo predictivo no desarrollara sesgos hacia las clases mayoritarias.

Figura 46

Exploración y distribución del Dataset



3.4.4.2 Implementación del preprocesamiento de datos. El preprocesamiento se implementó como una cadena de transformación orientada a convertir las capturas PCAP en vectores numéricos comparables por el modelo. Esta fase incluyó la depuración inicial de paquetes, la agrupación del tráfico por ventanas observadas desde una misma IP de destino, la extracción de características estadísticas y entrópicas, la normalización de las variables y el balanceo del conjunto de entrenamiento. Con ello, se consiguió una matriz de entrada consistente y alineada con el comportamiento que luego debía reproducir el controlador en tiempo real.

Limpieza de datos y depuración de valores. La etapa inicial del preprocesamiento se centró en leer las capturas, filtrar únicamente los paquetes que contenían capa IP válida y ordenar cronológicamente el tráfico. Este procedimiento permitió depurar información irrelevante y asegurar que los registros conservasen coherencia temporal, condición necesaria para la construcción posterior de ventanas deslizantes y para el cálculo de variables dependientes del tiempo.

Figura 47

Implementación de la limpieza de datos y depuración de valores

```
def process(self) -> pd.DataFrame:
    print(f"\nProcesando : {self.pcap_path.name}  →  '{self.label}'")
    try:
        packets = rdpcap(str(self.pcap_path))
    except Exception as e:
        print(f"  ❌ Error: {e}")
        return pd.DataFrame()

    # Filtrar y ordenar por timestamp
    ip_pkts = [p for p in packets if p.haslayer(IP)]
    ip_pkts.sort(key=lambda p: float(p.time))

    print(f"  Paquetes totales  : {len(packets):>8,}")
    print(f"  Paquetes IP       : {len(ip_pkts):>8,}")

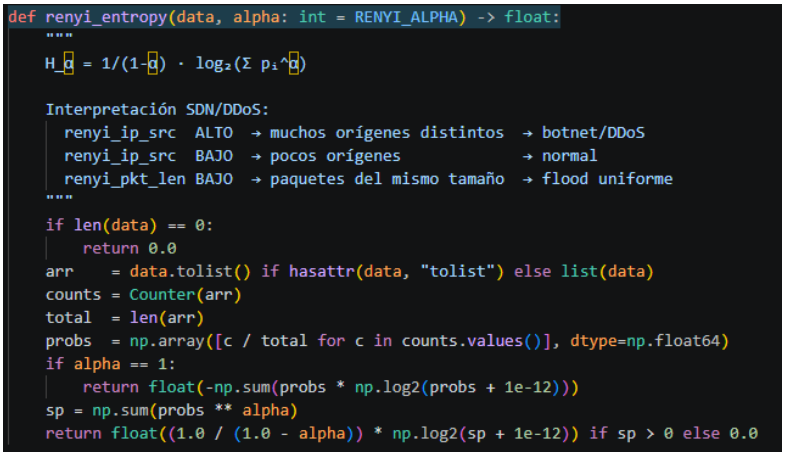
    # Agrupar por ip_dst
    groups = defaultdict(list)
    for pkt in ip_pkts:
        groups[pkt[IP].dst].append(pkt)
```

Este fragmento representa el inicio real del pipeline offline, donde el tráfico bruto se transforma en una colección ordenada de paquetes válidos, agrupados por destino y listos para el proceso de ingeniería de características.

Ingeniería de características: Entropía de Rényi. La ingeniería de características constituyó el núcleo analítico del pipeline. En esta etapa se calcularon métricas estadísticas convencionales, como volumen de bytes, tasas de paquetes y diversidad de orígenes, junto con variables de información basadas en entropía de Rényi y entropía de Shannon. La inclusión de la entropía de Rényi sobre IP de origen, IP de destino y tamaño de paquete permitió capturar alteraciones sutiles en la distribución del tráfico, aportando sensibilidad frente a patrones compatibles con ataques DDoS de baja intensidad.

Figura 48

Implementación de la entropía de Rényi



```
def renyi_entropy(data, alpha: int = RENYI_ALPHA) -> float:
    """
    H_α = 1/(1-α) · log₂(Σ pᵢ^α)

    Interpretación SDN/DDoS:
      renyi_ip_src  ALTO  → muchos orígenes distintos  → botnet/DDoS
      renyi_ip_src  BAJO  → pocos orígenes             → normal
      renyi_pkt_len BAJO  → paquetes del mismo tamaño  → flood uniforme
    """
    if len(data) == 0:
        return 0.0
    arr    = data.tolist() if hasattr(data, "tolist") else list(data)
    counts = Counter(arr)
    total  = len(arr)
    probs  = np.array([c / total for c in counts.values()], dtype=np.float64)
    if alpha == 1:
        return float(-np.sum(probs * np.log2(probs + 1e-12)))
    sp = np.sum(probs ** alpha)
    return float((1.0 / (1.0 - alpha)) * np.log2(sp + 1e-12)) if sp > 0 else 0.0
```

El primer bloque define la función matemática empleada para cuantificar diversidad y concentración en el tráfico, mientras que el segundo bloque que se muestra en la siguiente Figura 49 se denota cómo dicha medida se integra junto con otras variables dentro del vector de características que alimenta al modelo.

Figura 49

Implementación de la Entropía de Rényi mediante la integración con el vector de características

```
return {
    # Tiempo
    "duration":              duration,
    "iat_mean":              float(np.mean(iat)),
    "iat_std":               float(np.std(iat)),
    "iat_max":               float(np.max(iat)),
    "iat_min":               float(np.min(iat)),
    # Tamaño
    "packet_count":          int(len(sizes)),
    "total_bytes":           float(np.sum(sizes)),
    "pkt_len_mean":          float(np.mean(sizes)),
    "pkt_len_std":           float(np.std(sizes)),
    "pkt_len_max":           float(np.max(sizes)),
    "pkt_len_min":           float(np.min(sizes)),
    "pkt_len_var":           float(np.var(sizes)),
    # Tasas
    "pkt_rate":              len(sizes) / max(duration, 0.001),
    "byte_rate":             float(np.sum(sizes)) / max(duration, 0.001),
    # Protocolos
    "tcp_ratio":             pc.get("TCP",  0) / total,
    "udp_ratio":             pc.get("UDP",  0) / total,
    "icmp_ratio":            pc.get("ICMP", 0) / total,
    # Flags TCP
    "syn_count":             int(np.sum(flags == 2)),
    "ack_count":             int(np.sum(flags == 16)),
    "fin_count":             int(np.sum(flags == 1)),
    "rst_count":             int(np.sum(flags == 4)),
    # Diversidad de fuentes ← features SDN clave
    "unique_ips_src":        unique_ips_src,
    "unique_ports_src":      len(set(ports_src.tolist())),
    "unique_ports_dst":      len(set(ports_dst.tolist())),
    "src_diversity_ratio":   unique_ips_src / max(len(ips_src), 1),
    "ip_src_concentration":  top_src_count / len(ips_src),
    # ── ENTROPÍAS DE RÉNYI (α=3) ── espejadas en Ryu ──────────────
    "renyi_ip_src":          renyi_entropy(ips_src,  alpha=RENYI_ALPHA),
    "renyi_ip_dst":          renyi_entropy(
                                 np.array([dst_int] * len(sizes)),
                                 alpha=RENYI_ALPHA),
    "renyi_pkt_len":         renyi_entropy(sizes,    alpha=RENYI_ALPHA),
    # Entropías Shannon complementarias
    "shannon_ip_src":        shannon_entropy(ips_src),
    "shannon_port_src":      shannon_entropy(ports_src),
    "shannon_port_dst":      shannon_entropy(ports_dst),
}
```

Normalización y transformación de datos. Se transformaron las etiquetas categóricas a numéricas y se estandarizaron las variables continuas tras construir la matriz de características. Esta etapa fue necesaria porque el conjunto de datos contenía atributos con escalas muy distintas, por lo que la aplicación de StandardScaler permitió homogeneizar la distribución numérica de las variables y evitar sesgos durante el entrenamiento del clasificador. Asimismo, la partición estratificada del dataset garantizó una distribución estable de clases en los subconjuntos derivados.

Figura 50

Implementación de la normalización y transformación de datos



```python
def prepare_data(self, df: pd.DataFrame):

    self.feature_names = [c for c in df.columns if c != "label"]
    X     = df[self.feature_names].values.astype(np.float64)
    y     = df["label"].values
    y_enc = self.label_encoder.fit_transform(y)
    self.class_names = self.label_encoder.classes_

    print(f"\n  Features       : {len(self.feature_names)}")
    print(f"  Clases         : {list(self.class_names)}")
    print(f"  Total muestras : {len(X):,}")
    self._print_dist(y_enc, "Distribución ORIGINAL")

    # ── Split ANTES del balanceo → test refleja distribución real ──────
    X_tr, X_tmp, y_tr, y_tmp = train_test_split(
        X, y_enc, test_size=0.30,
        random_state=self.random_state, stratify=y_enc
    )
    X_val, X_te, y_val, y_te = train_test_split(
        X_tmp, y_tmp, test_size=0.50,
        random_state=self.random_state, stratify=y_tmp
    )

    n = len(X)
    print(f"\n  Split 70/15/15 (pre-balanceo):")
    print(f"    Train : {len(X_tr):>7,}  ({len(X_tr)/n*100:.1f}%)")
    print(f"    Val   : {len(X_val):>7,}  ({len(X_val)/n*100:.1f}%)")
    print(f"    Test  : {len(X_te):>7,}  ({len(X_te)/n*100:.1f}%)")

    # ── Balanceo SOLO en TRAIN ──────────────────────────────────────────
    X_tr, y_tr = self._balance(X_tr, y_tr)
    self._print_dist(y_tr, "Distribución TRAIN post-balanceo")

    # ── Escalado (fit SOLO en train) ────────────────────────────────────
    X_tr_s  = self.scaler.fit_transform(X_tr)
    X_val_s = self.scaler.transform(X_val)
    X_te_s  = self.scaler.transform(X_te)

    print(f"\n  Split final:")
    print(f"    Train : {len(X_tr_s):>7,}")
    print(f"    Val   : {len(X_val_s):>7,}")
    print(f"    Test  : {len(X_te_s):>7,}")

    return (X_tr_s, y_tr), (X_val_s, y_val), (X_te_s, y_te)
```

Este bloque resume tres decisiones operativas clave como la selección de variables predictoras, codificación de etiquetas y normalización del dataset antes del entrenamiento.

Balanceo con SMOTE y UnderSampler. Debido al desbalance natural entre tráfico normal y clases de ataque, se implementó una estrategia híbrida de balanceo aplicada exclusivamente sobre el conjunto de entrenamiento. En una primera fase se utilizó SMOTE para incrementar de forma sintética las clases minoritarias; posteriormente, se aplicó RandomUnderSampler para reducir la sobrerrepresentación de la clase mayoritaria. Este procedimiento permitió estabilizar la

distribución de las categorías sin contaminar los conjuntos de validación y prueba, preservando así la validez del proceso experimental.

Figura 51

Implementación del balanceo con SMOTE y UnderSampler

```
def _balance(self, X, y):
    Fallback gaussiano si imbalanced-learn no está disponible.
    """
    unique, counts = np.unique(y, return_counts=True)
    print(f"\n  Balanceo SMOTE + UnderSampler:")
    print(f"    TARGET_MINORITY = {TARGET_MINORITY:,}")
    print(f"    TARGET_MAJORITY = {TARGET_MAJORITY:,}")

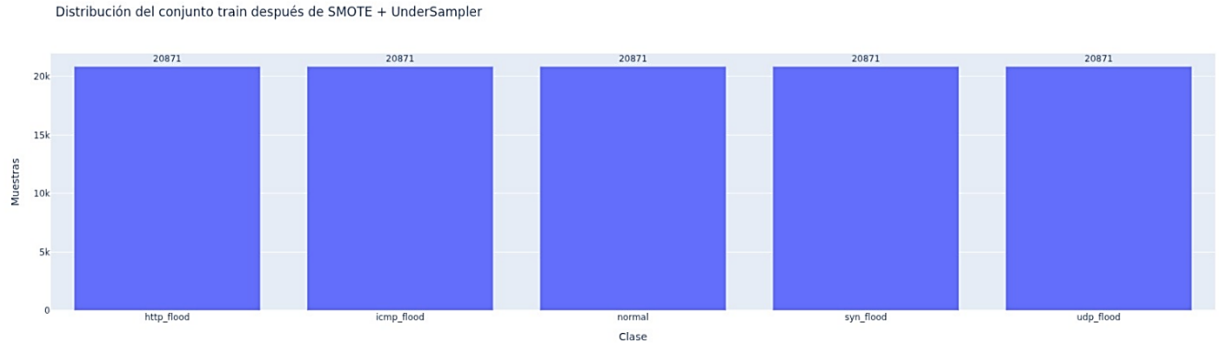
    if IMBLEARN_OK:
        # ── SMOTE ──────────────────────────────────────────────
        over_strategy = {
            int(cls): TARGET_MINORITY
            for cls, cnt in zip(unique, counts)
            if cnt < TARGET_MINORITY
        }
        if over_strategy:
            min_cnt = int(counts.min())
            k = min(5, max(1, min_cnt - 1))
            smote = SMOTE(
                sampling_strategy=over_strategy,
                k_neighbors=k,
                random_state=self.random_state
            )
            X, y = smote.fit_resample(X, y)
            print(f"    SMOTE  → {len(X):>8,} muestras  "
                  f"(clases subidas: {sorted(over_strategy.keys())})")

        # ── RandomUnderSampler ─────────────────────────────────
        u2, c2 = np.unique(y, return_counts=True)
        under_strategy = {
            int(cls): TARGET_MAJORITY
            for cls, cnt in zip(u2, c2)
            if cnt > TARGET_MAJORITY
        }
        if under_strategy:
            rus = RandomUnderSampler(
                sampling_strategy=under_strategy,
                random_state=self.random_state
            )
            X, y = rus.fit_resample(X, y)
            print(f"    Under  → {len(X):>8,} muestras  "
                  f"(clases bajadas: {sorted(under_strategy.keys())})")
```

Este fragmento evidencia la aplicación secuencial de sobremuestreo y submuestreo, reforzando la capacidad del modelo para aprender patrones minoritarios sin quedar dominado por la clase normal.

Figura 52

Visualización del Balanceo de Clases

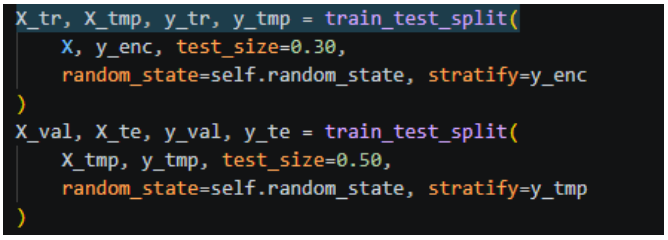


3.4.4.3 Implementación del entrenamiento del modelo. La fase de entrenamiento tuvo como objetivo construir un clasificador multiclase robusto, compatible con el entorno Ryu y capaz de generalizar frente a tráfico no visto. Para ello, el dataset ya preprocesado se dividió en subconjuntos de entrenamiento, validación y prueba, se configuró un modelo XGBoost con criterios de regularización y control de complejidad, y finalmente se exportaron todos los artefactos requeridos para su despliegue operativo en el controlador.

División de los datos para entrenamiento, validación y prueba. La partición del conjunto de datos se implementó bajo un esquema 70/15/15, manteniendo la proporción de clases mediante estratificación. Esta división permitió entrenar el clasificador sobre un bloque principal, monitorear su ajuste fino mediante validación y conservar un subconjunto independiente para la evaluación objetiva del desempeño.

Figura 53

Implementación de la división de datos para entrenamiento, validación y prueba



```
X_tr, X_tmp, y_tr, y_tmp = train_test_split(
    X, y_enc, test_size=0.30,
    random_state=self.random_state, stratify=y_enc
)
X_val, X_te, y_val, y_te = train_test_split(
    X_tmp, y_tmp, test_size=0.50,
    random_state=self.random_state, stratify=y_tmp
)
```

Este bloque materializa la estrategia de partición definida en el diseño metodológico y garantiza la separación adecuada entre las fases de aprendizaje, validación y prueba

Entrenamiento del modelo XGBoost. La implementación se llevó a cabo a partir del clasificador XGBClassifier, utilizando un conjunto de parámetros que configuraba qué condiciones deben cumplirse para un adecuado ajustado del modelo, mejorar la capacidad de generalización del modelo y mitigar el overfitting. Como parámetros se definieron la profundidad máxima, la tasa de aprendizaje, la regularización, el submuestreo y un método de early stopping sobre el conjunto de validación.

Este esquema permitió el ajuste del clasificador dentro de un entorno controlado antes de ser incrustado en el plano de control de la red. Esto se lo puede llegar a visualizar de mejor manera en la siguiente Figura 54, en donde se detalla el código con los diferentes hiperparámetros necesarios para el entrenamiento adecuado para el modelo planteado.

Figura 54

Implementación del entrenamiento del modelo XGBoost

```
params = {
    "objective":              "multi:softprob",
    "num_class":              len(self.class_names),
    "eval_metric":            "mlogloss",
    # Capacidad
    "max_depth":              4,
    "min_child_weight":       8,
    "gamma":                  0.4,
    # Boosting
    "n_estimators":           1500,
    "learning_rate":          0.1,
    "early_stopping_rounds":  50,
    # Regularización (anti-overfitting)
    "subsample":              0.75,
    "colsample_bytree":       0.75,
    "colsample_bylevel":      0.75,   # ← del código nuevo
    "reg_alpha":              0.8,
    "reg_lambda":             2.5,
    "random_state":           self.random_state,
    "n_jobs":                 -1,
}

print(f"\n  Parámetros clave:")
key_params = ["max_depth", "learning_rate", "n_estimators",
              "early_stopping_rounds", "reg_alpha", "reg_lambda",
              "subsample", "colsample_bytree", "colsample_bylevel"]
for k in key_params:
    print(f"    {k:<30} = {params[k]}")

self.model = xgb.XGBClassifier(**params)
self.model.fit(
    X_tr, y_tr,
    sample_weight = sw,
    eval_set      = [(X_tr, y_tr), (X_val, y_val)],
    verbose       = 100,
)
```

El fragmento anterior representa la fase central de aprendizaje del clasificador, donde el modelo ajusta sus parámetros internos a partir de las características extraídas del tráfico de red.

Figura 55

Generación de Gráficas del modelo



```
print("\n  1. CLASSIFICATION REPORT\n")
print(report)
(save_dir / "classification_report.txt").write_text(report, encoding="utf-8")

acc = accuracy_score(y_test, y_pred)
prec, rec, f1, _ = precision_recall_fscore_support(
    y_test, y_pred, average="weighted", zero_division=0)
y_bin     = label_binarize(y_test, classes=list(range(len(self.class_names))))
auc_macro = roc_auc_score(y_bin, y_pred_proba,
                          multi_class="ovr", average="macro")

print("  2. MÉTRICAS GLOBALES\n")
print(f"    {'Accuracy':<25} {acc:.4f}")
print(f"    {'Precision (weighted)':<25} {prec:.4f}")
print(f"    {'Recall (weighted)':<25} {rec:.4f}")
print(f"    {'F1-Score (weighted)':<25} {f1:.4f}")
print(f"    {'ROC-AUC (macro OvR)':<25} {auc_macro:.4f}")

with open(save_dir / "metrics.json", "w") as fh:
    json.dump({
        "accuracy":           round(float(acc),      4),
        "precision_weighted": round(float(prec),     4),
        "recall_weighted":    round(float(rec),      4),
        "f1_weighted":        round(float(f1),       4),
        "roc_auc_macro":      round(float(auc_macro),4),
        "best_iteration":     int(self.model.best_iteration),
        "best_mlogloss_val":  round(float(self.model.best_score), 4),
    }, fh, indent=2)

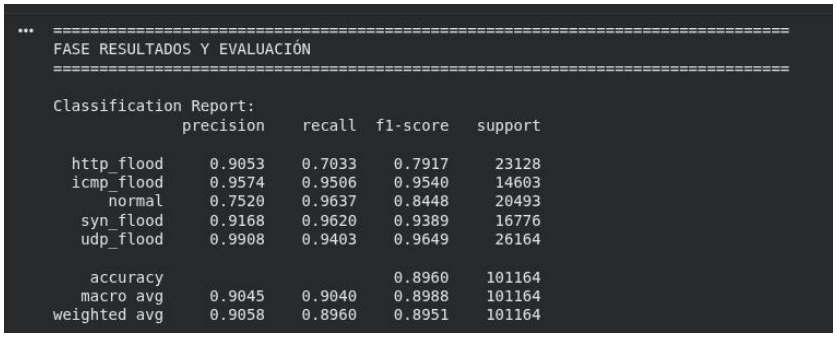
print("\n  3. GRÁFICAS\n")
self._plot_confusion_matrix(y_test, y_pred, save_dir)
self._plot_learning_curves(save_dir)
```

Estimación del rendimiento del Modelo XGBoost. Tras la fase predictiva, se generó un informe de clasificación detallado y se calcularon las métricas globales exclusivamente sobre el conjunto de prueba 15% de los datos aislados, lo que proporcionó una visión exhaustiva del rendimiento del modelo en términos de exactitud (accuracy), precisión, sensibilidad (recall), puntuación F1-score ponderada y el Área Bajo la Curva (ROC-AUC) multiclase.

El modelo XGBoost, entrenado con los mejores hiperparámetros para salvar el sobreajuste, mostró un rendimiento extremadamente eficaz y equilibrado. A nivel global, el algoritmo alcanzó una impresionante exactitud, mostrando que la inmensa mayoría de los flujos de red fueron clasificados correctamente en la clase correspondiente. En el análisis por clases, el modelo mostró una precisión y un recall muy altos para la clase traffic benign legit y de ataque masivo traffic flood UDP y SYN Flood, mostrando de esta forma una gran capacidad para manejar el tráfico más masivo generado sin generar falsos positivos que puedan afectar la disponibilidad de la red.

Figura 56

Resultado de las predicciones del Modelo en el conjunto de prueba



```
================================================================================
FASE RESULTADOS Y EVALUACIÓN
================================================================================

Classification Report:
              precision    recall  f1-score   support

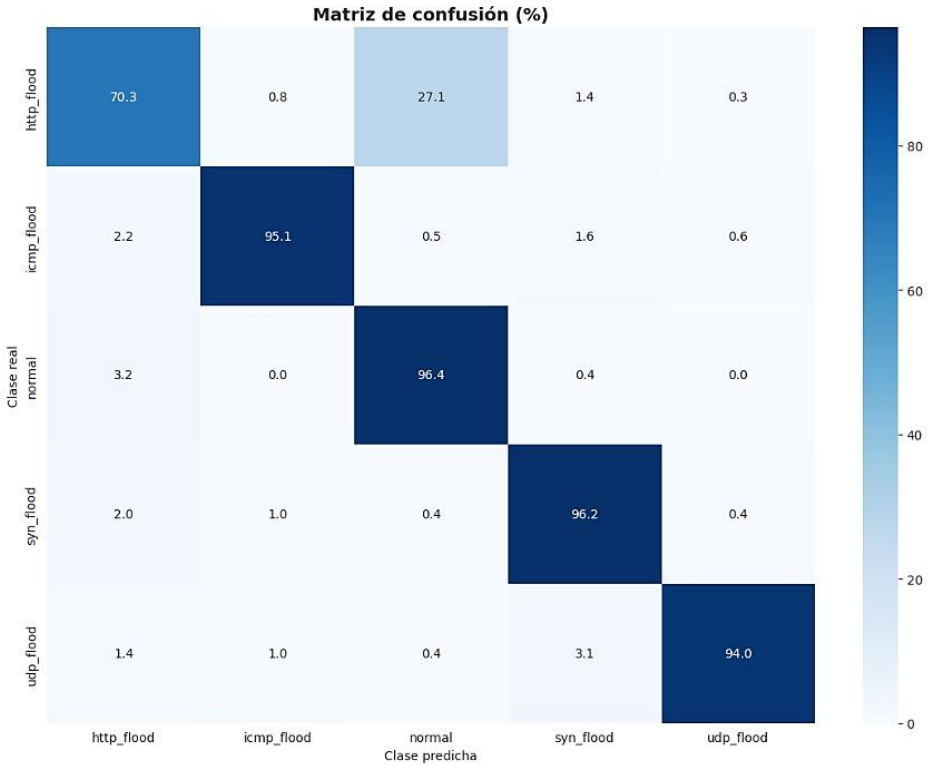
  http_flood     0.9053    0.7033    0.7917     23128
  icmp_flood     0.9574    0.9506    0.9540     14603
      normal     0.7520    0.9637    0.8448     20493
   syn_flood     0.9168    0.9620    0.9389     16776
   udp_flood     0.9908    0.9403    0.9649     26164

    accuracy                         0.8960    101164
   macro avg     0.9045    0.9040    0.8988    101164
weighted avg     0.9058    0.8960    0.8951    101164
```

Matriz de Confusión del modelo XGBoost. La visualización de las matrices de confusión sobre el conjunto de prueba aislado permitió evaluar de forma empírica y detallada la calidad de las predicciones del modelo XGBoost multiclase, al comparar directamente las etiquetas predichas por el algoritmo con las etiquetas reales del tráfico. Cada celda de la matriz indica la proporción e instancias clasificadas correcta o incorrectamente para cada categoría de red. La diagonal principal representa las predicciones correctas verdaderos positivos de cada clase, mientras que las celdas fuera de esta diagonal exponen los errores de clasificación y revelan confusiones específicas entre los distintos tipos de ataques.

Figura 57

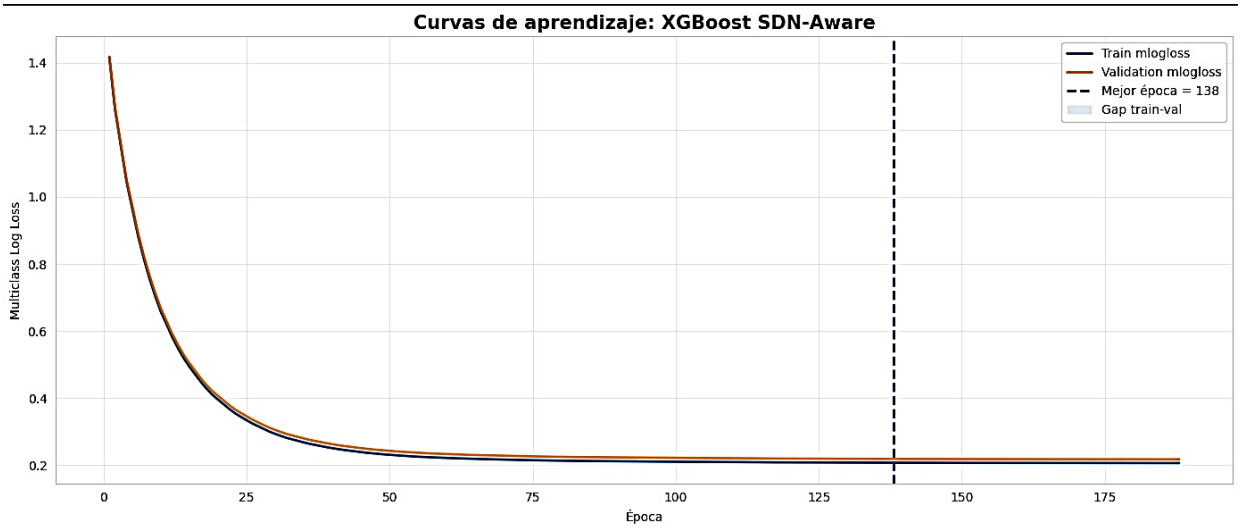
Matriz de confusión con los datos de test



Auc ROC y Loss Funnction del modelo XGBoost. Para evaluar exhaustivamente la capacidad de separación espacial y la convergencia del modelo XGBoost, se procedió a calcular y graficar tanto las curvas ROC Receiver Operating Characteristic como la evolución de la función de pérdida Loss Function. Dado que el sistema aborda un problema de detección multiclase seis categorías de tráfico, la evaluación ROC se implementó bajo la estrategia de Uno-contra-Resto (OvR, One-vs-Rest). Para ello, se obtuvieron las probabilidades predictivas para cada instancia del conjunto de prueba utilizando el método predict_proba. Posteriormente, se calcularon la Tasa de Falsos Positivos (FPR) y la Tasa de Verdaderos Positivos (TPR) individuales, permitiendo generar la gráfica ROC y determinar el Área Bajo la Curva (AUC) con un enfoque macro (el promedio de todas las clases).

Figura 58

Curva logarítmica Multiclase

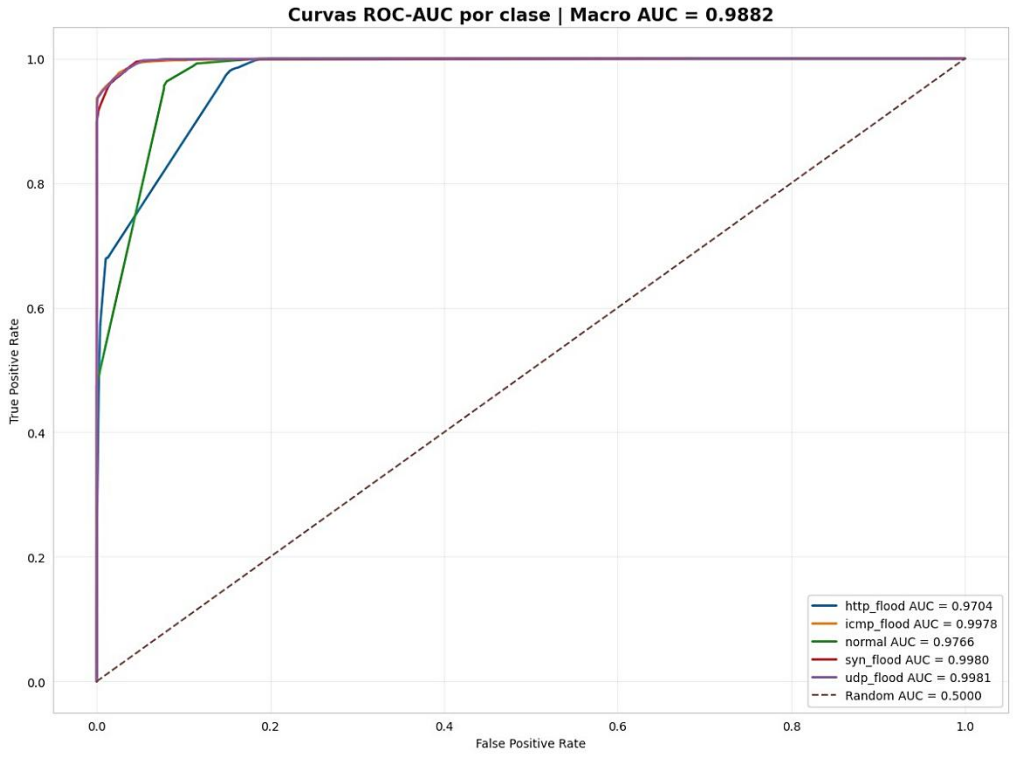


Se evaluó la estabilidad del aprendizaje mediante el análisis de la función de pérdida logarítmica multiclase mlogloss. La gráfica de las curvas de aprendizaje Learning Curves permitió monitorear la evolución del error de predicción tanto en el subconjunto de entrenamiento como en el de validación a lo largo de las iteraciones (epochs).

La convergencia de las curvas de entrenamiento y validación indica un aprendizaje estable sin sobreajuste. Además, esta visualización ilustró el comportamiento del mecanismo de Parada Temprana, marcando verticalmente la iteración exacta donde el error de validación alcanzó su punto mínimo global antes de empezar a degradarse. Este análisis garantizó que el modelo consolidado operara en su punto máximo de eficiencia algorítmica.

Figura 59

Área bajo la curva ROC (AUC) del rendimiento del modelo



La visualización de las curvas ROC constituye una herramienta fundamental para validar la eficacia del modelo en la separación de las amenazas:

- **Eje X (Tasa de Falsos Positivos - FPR):** Representó la proporción de flujos de red que fueron clasificados incorrectamente como pertenecientes a un ataque específico. Un FPR bajo indicó la robustez del modelo para no generar falsas alarmas (falsos positivos) que pudieran afectar el tráfico legítimo.
- **Eje Y (Tasa de Verdaderos Positivos - TPR):** Representó la sensibilidad del clasificador; es decir, la proporción de flujos de un ataque real que fueron correctamente identificados y mitigados.

- **Línea Diagonal Discontinua:** Esta línea representó el comportamiento estadístico de un clasificador aleatorio (AUC = 0.50). El objetivo de cualquier modelo predictivo es situar sus curvas lo más alejadas y por encima posible de esta referencia empírica.
- **Curvas Multiclase:** Cada curva de color representó el rendimiento de detección aislado para cada categoría. A medida que las curvas se acercaron al vértice superior izquierdo del gráfico donde TPR es 1.0 y FPR es 0.0, se confirmó la alta precisión del algoritmo.
- **Área Bajo la Curva (Macro AUC):** El AUC proporcionó un valor escalar que resume el rendimiento global del gráfico. Al promediar el desempeño frente a todas las variantes de ataques, un valor Macro AUC cercano a 1.0 garantizó que el XGBoost posee una capacidad analítica excelente para distinguir las anomalías del tráfico benigno

Exportación del modelo para su integración con Ryu. Como parte final de esta fase se exportaron el clasificador entrenado, el escalador, el codificador de etiquetas y la metadata del pipeline a un directorio estructurado para despliegue. Esta decisión fue esencial para garantizar que el controlador Ryu reconstruyera en línea el mismo esquema de inferencia definido durante el entrenamiento offline, manteniendo coherencia entre ambos entornos.

Figura 60

Implementación de la exportación del modelo



```python
def export_for_ryu(self, export_dir: str):
    ep = Path(export_dir)
    ep.mkdir(parents=True, exist_ok=True)

    self.model.get_booster().save_model(str(ep / "xgboost_model.ubj"))
    joblib.dump(self.scaler,        ep / "scaler.joblib")
    joblib.dump(self.label_encoder, ep / "label_encoder.joblib")

    with open(ep / "feature_names.json", "w", encoding="utf-8") as f:
        json.dump(self.feature_names, f, ensure_ascii=False, indent=2)

    meta = {
        "classes":             list(self.class_names),
        "num_classes":         int(len(self.class_names)),
        "window_mode":         "per_ip_dst",
        "window_size":         WINDOW_SIZE,
        "window_step":         WINDOW_STEP,
        "renyi_alpha":         RENYI_ALPHA,
        "renyi_features":      ["renyi_ip_src", "renyi_ip_dst", "renyi_pkt_len"],
        "shannon_features":    ["shannon_ip_src", "shannon_port_src", "shannon_port_dst"],
        "sdn_features":        ["src_diversity_ratio", "ip_src_concentration",
                                "unique_ips_src"],
        "split":               "70/15/15",
        "balancing":           "SMOTE + RandomUnderSampler (train only)",
        "model_file":          "xgboost_model.ubj",
        "scaler_file":         "scaler.joblib",
        "label_encoder_file":  "label_encoder.joblib",
        "feature_names_file":  "feature_names.json",
        "best_iteration":      int(self.model.best_iteration),
    }
    with open(ep / "meta.json", "w", encoding="utf-8") as f:
        json.dump(meta, f, ensure_ascii=False, indent=2)
```

Para evidenciar la continuidad entre entrenamiento offline e inferencia online, puede añadirse un fragmento breve del controlador donde se cargan dichos artefactos. Esto demuestra que el pipeline no concluye en el entrenamiento, sino que se proyecta hacia la operación real del sistema de detección.

Figura 61

Implementación de la integración con Ryu



```python
class MLInference:
    def __init__(self, model_dir: Path):
        booster = xgb.Booster()
        booster.load_model(str(model_dir / "xgboost_model.ubj"))
        self.model = booster
        self.scaler = joblib.load(model_dir / "scaler.joblib")
        self.label_encoder = joblib.load(model_dir / "label_encoder.joblib")

        with open(model_dir / "feature_names.json", "r", encoding="utf-8") as f:
            self.feature_names = json.load(f)

        with open(model_dir / "meta.json", "r", encoding="utf-8") as f:
            self.meta = json.load(f)
```

3.4.4.4 Implementación del Dashboard de administración de atacantes. El dashboard de administración de atacantes se implementó como una interfaz de control operativo para gestionar, desde un punto centralizado, la activación de nodos maliciosos y la ejecución de campañas de tráfico hostil dentro del laboratorio SDN. A diferencia del dashboard de visualización del controlador, este componente realiza un papel activo en el entorno experimental ya que permite levantar bots por medio de contenedores Docker al igual que controlar su estado y emitir comandos de ataque, elegir el tipo de tráfico malicioso, determinar la intensidad en el número de paquetes y el objetivo de la campaña. Su implementación se llevó a cabo mediante Flask como servidor web y Docker SDK para la gestión programática de los contenedores.

En primer lugar, lo que se lleva a cabo es la verificación de la existencia del motor Docker junto con el establecimiento del cliente del socket de la conexión actual con el socket local. Esta verificación, en el fondo, es una manera de comprobar que se está listo para levantar los contenedores que llevarán a cabo la funcionalidad de bots atacantes. Si no hay conexión con Docker, se registraría la ocurrencia de un error en el sistema y se evitaría ejecutar la parte del sistema que depende del levantamiento de contenedores.

Figura 62

Implementación del Dashboard de administración de atacantes - verificación del motor Docker

```
try:
    import docker as docker_sdk
    _DOCKER_SOCKET = "unix:///var/run/docker.sock"
    _docker_client = docker_sdk.DockerClient(base_url=_DOCKER_SOCKET)
    _docker_client.ping()
    DOCKER_OK = True
    print(f"[C2] Docker OK — socket: {_DOCKER_SOCKET}")
except Exception as _e:
    try:
        _docker_client = docker_sdk.from_env()
        _docker_client.ping()
        DOCKER_OK = True
    except Exception as _e2:
        DOCKER_OK = False
        _docker_client = None
        print(f"[C2] Docker no disponible: {_e2}")
```

El dashboard mantiene un estado global del laboratorio, donde se almacenan el comando actual, los bots registrados, el historial de acciones ejecutadas, la lista de contenedores gestionados y el número de bots solicitados. Esta estructura permite controlar el comportamiento del entorno de ataque, conservar trazabilidad de las campañas y reflejar en la interfaz el estado operativo de los bots.

Figura 63

Implementación del Dashboard de administración de atacantes - almacenamiento de bots y contenedores

```
state = {
    "command":    None,
    "issued_at":  None,
    "bots":       {},
    "history":    [],
    "containers": [],
    "bot_count":  0,
}

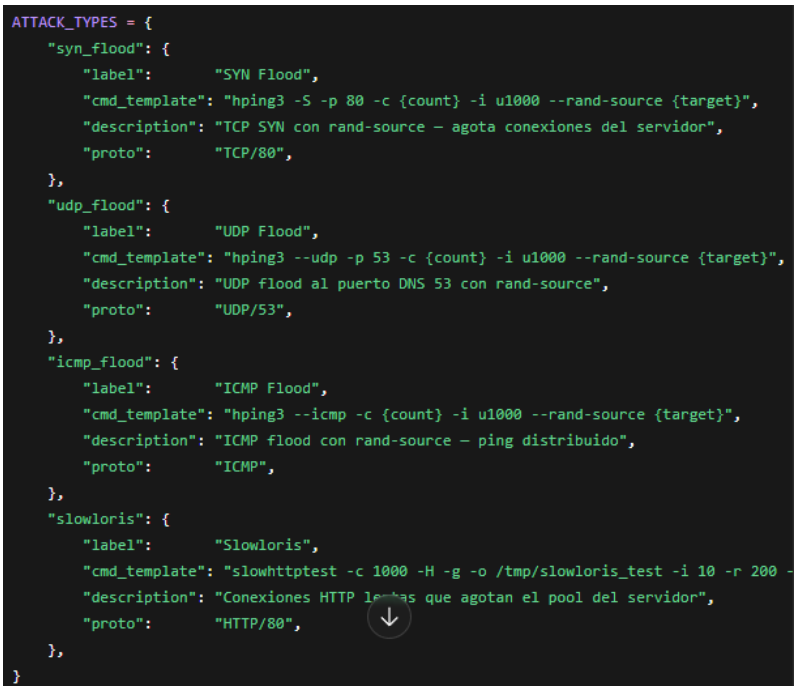
BOT_IMAGE       = "ddos-bot"
BOT_NAME_PREFIX = "c2-bot"
C2_SELF_URL     = os.environ.get("C2_URL", "http://192.168.56.100:8888")

INTENSIDADES = [50, 75, 100, 200]
INTENSIDAD_DEFAULT = 75
```

Posteriormente, se definió un catálogo de tipos de ataque disponibles, donde cada vector malicioso se asocia con una etiqueta, una plantilla de comando, una descripción funcional y el protocolo afectado. Esta estructura permite parametrizar ataques como SYN Flood, UDP Flood e ICMP Flood con herramientas como hping3 y slowhttptest. De esta manera, el operador no ejecuta comandos manualmente en cada bot, sino que selecciona el tipo de ataque desde el dashboard y el sistema genera el comando correspondiente de forma automática.

Figura 64

Implementación del Dashboard de administración de atacantes - definición de tipos de ataques disponibles



Para mantener el control de los bots activos, se implementó una API de registro mediante el endpoint /heartbeat. Cada bot envía periódicamente su identificador, dirección IP y estado actual, permitiendo al dashboard distinguir entre bots inactivos, activos, atacando o desconectados. Esta lógica resulta importante porque permite verificar cuántos atacantes se encuentran realmente disponibles antes de iniciar una campaña.

Figura 65

Implementación del Dashboard de administración de atacantes - API de registro



```python
@app.route("/heartbeat", methods=["POST"])
def heartbeat():
    data = request.json or {}
    bot_id  = data.get("bot_id", str(uuid.uuid4())[:8])
    bot_ip  = request.remote_addr
    status  = data.get("status", "idle")

    with lock:
        if bot_id not in state["bots"]:
            state["bots"][bot_id] = {
                "ip": bot_ip,
                "last_seen": time.time(),
                "status": status,
                "executions": 0,
                "first_seen": datetime.utcnow().isoformat(),
            }
        else:
            state["bots"][bot_id]["last_seen"] = time.time()
            state["bots"][bot_id]["status"] = status
            if status == "attacking":
                state["bots"][bot_id]["executions"] += 1

    return jsonify({"ok": True, "bot_id": bot_id})
```

La emisión de comandos se realizó mediante el endpoint /api/issue, encargado de recibir desde el dashboard el tipo de ataque, la dirección objetivo y la intensidad seleccionada. Con estos datos, el sistema valida el ataque solicitado, construye el comando de ejecución a partir de la plantilla definida y lo almacena como comando vigente para que los bots lo consulten y ejecuten. Además, registra la acción en el historial y actualiza el estado de los bots para reflejar visualmente que se encuentran atacando.

Figura 66

Implementación del Dashboard de administración de atacantes - registro de la acción en historial y actualización de estado de los bots



```python
@app.route("/api/issue", methods=["POST"])
def issue_command():
    data       = request.json or {}
    attack     = data.get("attack_type", "stop")
    target     = data.get("target", "192.168.56.20")
    port       = "predefinido"
    duration   = 60

    if attack not in ATTACK_TYPES:
        return jsonify({"error": "tipo de ataque desconocido"}), 400

    atk_info = ATTACK_TYPES[attack]
    count = int(data.get("count", INTENSIDAD_DEFAULT))
    if count not in INTENSIDADES:
        count = INTENSIDAD_DEFAULT

    if attack == "stop":
        cmd = {"type": "stop", "label": "DETENER TODO"}
    else:
        raw_cmd = atk_info["cmd_template"].format(
            target=target, port=port, count=count
        )
        shell = f"{raw_cmd} || true"
        cmd = {
            "type":      attack,
            "label":     atk_info["label"],
            "shell":     shell,
            "target":    target,
            "port":      port,
            "count":     count,
            "duration":  duration,
            "issued_at": datetime.utcnow().isoformat(),
        }

    with lock:
        state["command"] = cmd
        state["issued_at"] = time.time()
        state["history"].insert(0, {
            "ts":     datetime.utcnow().strftime("%H:%M:%S"),
            "type":   attack,
            "label":  atk_info["label"],
            "target": target,
            "port":   port,
            "count":  count if attack != "stop" else 0,
            "bots":   len(state["bots"]),
        })

    return jsonify({"ok": True, "command": cmd})
```

Adicionalmente, los bots consultan el comando vigente mediante el endpoint /cmd. Esta separación entre emisión y consulta permite que el dashboard funcione bajo una lógica de comando y control: el servidor central define la instrucción, mientras que cada bot la solicita periódicamente y la ejecuta dentro de su propio contenedor.

Figura 67

Implementación del Dashboard de administración de atacantes - acción de bots de consulta al endpoint

```
@app.route("/cmd", methods=["GET"])
def get_command():
    with lock:
        cmd = state["command"]
    return jsonify({"command": cmd})
```

Para el despliegue dinámico de atacantes, se implementó el endpoint /api/bots/spawn, encargado de levantar una cantidad configurable de contenedores Docker a partir de la imagen ddos-bot. Cada contenedor opera de forma desacoplada, utiliza la red del host y recibe variables de entorno como la URL del servidor C2 y su identificador de bot. Esta configuración facilita que los atacantes se integren rápidamente al laboratorio sin requerir creación manual de procesos o máquinas adicionales.

Figura 68

Implementación del Dashboard de administración de atacantes - despliegue dinámico de los atacantes

```python
@app.route("/api/bots/spawn", methods=["POST"])
def spawn_bots():
    if not DOCKER_OK:
        return jsonify({"error": "Docker no disponible — instala: pip install docker"}), 503

    data  = request.json or {}
    count = max(1, min(int(data.get("count", 5)), 50))

    launched = []
    errors   = []

    for i in range(1, count + 1):
        name = f"{BOT_NAME_PREFIX}-{uuid.uuid4().hex[:6]}"
        try:
            container = _docker_client.containers.run(
                BOT_IMAGE,
                name=name,
                detach=True,
                remove=True,
                network_mode="host",
                cap_add=["NET_ADMIN", "NET_RAW"],
                environment={
                    "C2_URL": C2_SELF_URL,
                    "BOT_ID": f"bot-{name[-6:]}",
                },
            )
            launched.append(container.id[:12])
            with lock:
                state["containers"].append(container.id)
        except Exception as e:
            errors.append(str(e))

    with lock:
        state["bot_count"] = len(state["containers"])

    return jsonify({
        "ok":       len(launched) > 0,
        "launched": len(launched),
        "errors":   errors,
        "ids":      launched,
    })
```

Finalmente, se implementó una función de detención general mediante /api/bots/stop_all, la cual permite detener y eliminar todos los contenedores asociados al laboratorio. Esta función es necesaria para cerrar campañas de ataque, limpiar el entorno experimental y evitar que procesos maliciosos de prueba continúen activos después de cada sesión.

Figura 69

Implementación del Dashboard de administración de atacantes - función de detención general

```python
@app.route("/api/bots/stop_all", methods=["POST"])
def stop_all_bots():
    if not DOCKER_OK:
        return jsonify({"error": "Docker no disponible"}), 503

    stopped = 0
    errors  = []

    all_bots = _docker_client.containers.list(
        all=True, filters={"name": BOT_NAME_PREFIX}
    )

    for c in all_bots:
        try:
            c.stop(timeout=2)
            stopped += 1
        except docker_sdk.errors.NotFound:
            stopped += 1
        except Exception:
            try:
                c.kill()
                stopped += 1
            except Exception as e2:
                errors.append(f"{c.name}: {e2}")

    with lock:
        state["containers"] = []
        state["bot_count"]  = 0
        state["bots"]       = {}

    return jsonify({"ok": True, "stopped": stopped, "errors": errors})
```

En conjunto, esta implementación permitió que el dashboard de administración de atacantes funcionara como una capa de orquestación experimental para controlar la cantidad de atacantes, el tipo de ataque, la intensidad, el objetivo y la detención de campañas. De esta forma, el sistema proporcionó un mecanismo reproducible y controlado para generar tráfico malicioso dentro del entorno SDN, aportando consistencia a la construcción del dataset y a las posteriores pruebas de validación del sistema de detección. De forma visual se puede observar en las siguientes Figuras 70, 71 y 72 como quedaron plasmadas de acuerdo con el establecimiento en el diseño en el punto 3.3.5.2 con todas las configuraciones, requerimientos y división de pestañas tanto para atacantes como de usuarios normales.

Figura 70

Dashboard de administración de atacantes - primera pestaña

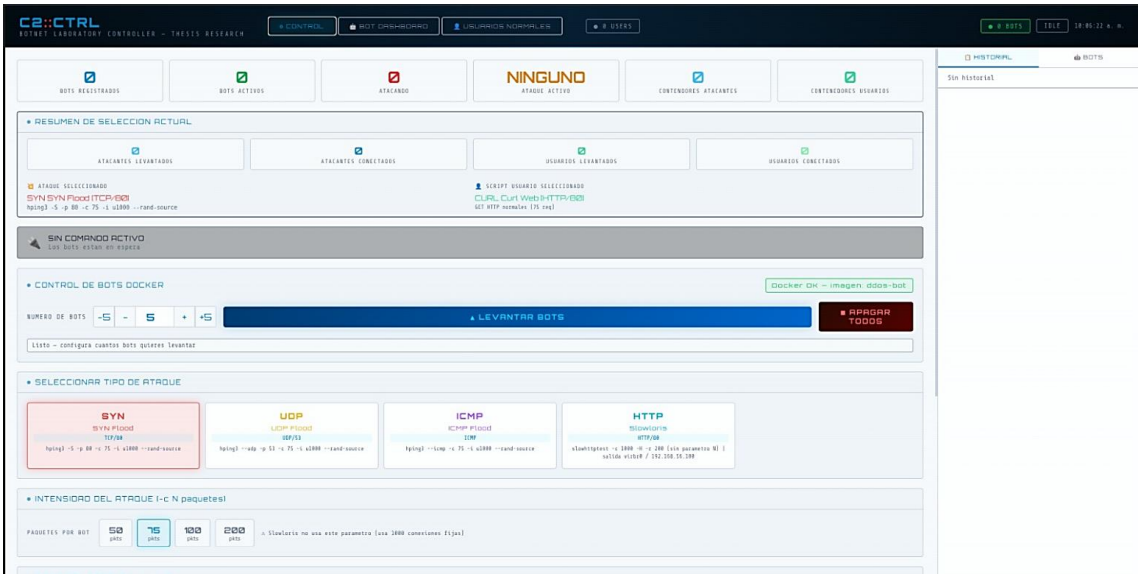


Figura 71

Dashboard de administración de atacantes - segunda pestaña

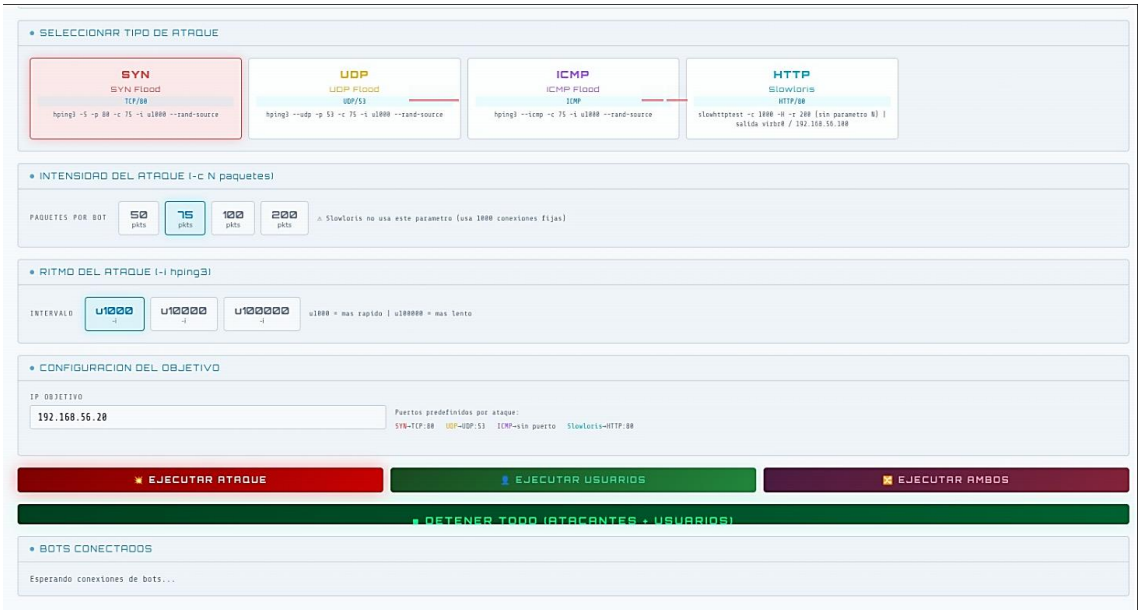
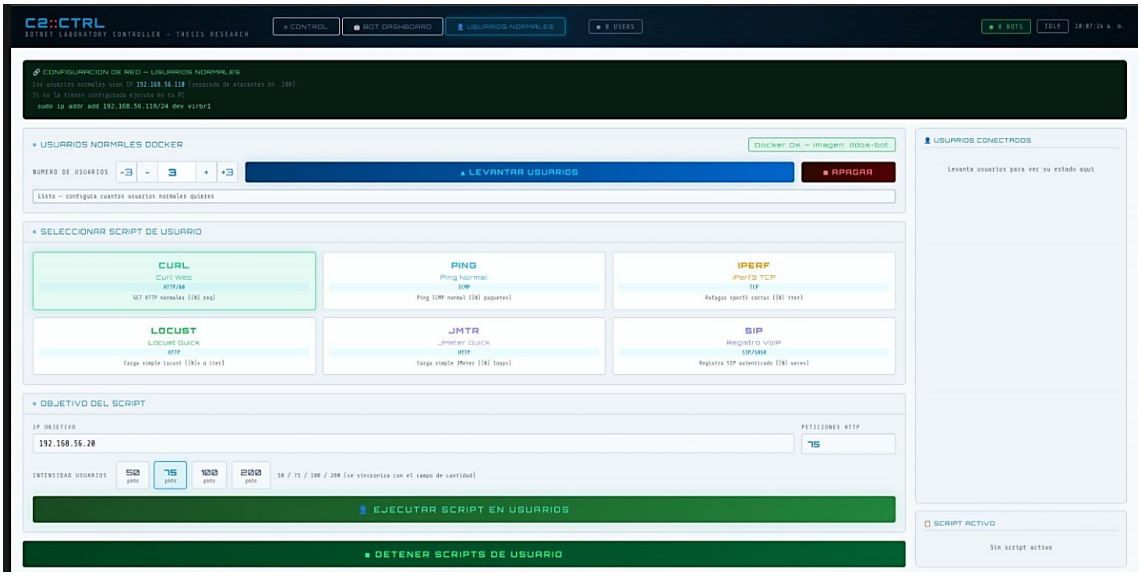


Figura 72

Dashboard de administración de atacantes - segunda parte de la segunda pestaña



3.4.5 Implementación del sistema del modelo interpretable SHAPASH

La implementación se realizó mediante un script en Python encargado de cargar el modelo entrenado, preparar el conjunto de datos y generar las explicaciones globales y locales con la biblioteca SHAPASH. El flujo implementado se organiza en las siguientes etapas.

3.4.5.1 Importación de bibliotecas. En primer lugar, se importan las bibliotecas necesarias para la lectura de archivos, manipulación de datos, carga del modelo y generación de explicaciones.

Figura 73

Implementación del sistema del modelo SHAPASH - importación de bibliotecas

```
import json
import joblib
import pandas as pd
import xgboost as xgb
from pathlib import Path
from shapash import SmartExplainer
```

Pandas manipula los datos, joblib recupera el escalador, xgboost carga el clasificador y SmartExplainer crea el componente interpretable.

3.4.5.2 Definición de rutas. Se establecen las rutas relativas donde se encuentra el conjunto de datos y los archivos exportados durante el entrenamiento.

Figura 74

Implementación del sistema del modelo SHAPASH - definición de rutas

```
base_dir = Path("data")
export_dir = Path("results2/ryu_export")
```

En la carpeta de exportación se almacenan el modelo XGBoost, el escalador y el listado de características utilizadas durante el entrenamiento.

3.4.5.3 Carga del modelo entrenado. El modelo se recupera desde el archivo xgboost_model.ubj como se lo puede observar en la siguiente Figura. Este procedimiento permite reutilizar el modelo previamente entrenado sin ejecutar nuevamente el proceso de ajuste.

Figura 75

Implementación del sistema del modelo SHAPASH – carga del modelo entrenado

```
model = xgb.XGBClassifier()
model.load_model(export_dir / "xgboost_model.ubj")
```

3.4.5.4 Carga del escalador. Posteriormente, se carga el objeto utilizado para normalizar las variables de entrada. Esto debido a que es indispensable utilizar el mismo escalador empleado durante el entrenamiento, ya que cualquier diferencia en la transformación de los datos produciría entradas incompatibles con el modelo.

Figura 76

Implementación del sistema del modelo SHAPASH - carga del escalador

```
scaler = joblib.load(export_dir / "scaler.joblib")
```

3.4.5.5 Recuperación de las características. El archivo feature_names.json contiene el nombre y el orden de las variables utilizadas por el clasificador. Debido a que el orden de estas características debe conservarse debido a que el modelo interpreta cada posición de entrada de acuerdo con la estructura utilizada durante su entrenamiento.

Figura 77

Implementación del sistema del modelo SHAPASH - recuperación de las características

```
with open(export_dir / "feature_names.json", "r") as f:
    feature_names = json.load(f)
```

3.4.5.6 Carga del conjunto de datos

Figura 78

Implementación del sistema del modelo SHAPASH - carga del conjunto de datos

```
df = pd.read_csv(base_dir / "dataset_combined.csv")
```

Como se pudo observar en la Figura 78 el archivo dataset_combined.csv es leído mediante pandas. Todo esto debido a que este conjunto contiene las observaciones normales y los registros correspondientes a los diferentes tipos de ataques de denegación de servicio.

3.4.5.7 Validación de la estructura de los datos. Antes de generar las explicaciones, el script verifica que exista la columna label y que todas las características requeridas por el modelo estén disponibles.

Figura 79

Implementación del sistema del modelo SHAPASH - validación de la estructura de los datos

```
if "label" not in df.columns:
    raise ValueError(
        "El archivo dataset_combined.csv no tiene la columna 'label'"
    )

missing_features = [
    col for col in feature_names
    if col not in df.columns
]

if missing_features:
    raise ValueError(
        f"Faltan estas features en el dataset: {missing_features}"
    )
```

Estas comprobaciones previenen la ejecución del explicador con datos incompletos o con una estructura distinta a la utilizada durante el entrenamiento.

3.4.5.8 Separación de variables predictoras y variable objetivo. Las variables de entrada se seleccionan empleando el orden almacenado en feature_names, mientras que la columna label se conserva como variable objetivo.

Figura 80

Implementación del sistema del modelo SHAPASH - separación ded variables predictoras y variable objetivo

```
X = df[feature_names].copy()
y_raw = df["label"].copy()
```

3.4.5.9 Conversión de etiquetas a valores numéricos. Las etiquetas textuales se convierten a códigos numéricos, tal cual como se puede denotar en la siguiente Figura 81.

Figura 81

Implementación del sistema del modelo SHAPASH - conversión de etiquetas a valores numéricos

```
label_map = {
    "normal": 0,
    "http_flood": 1,
    "udp_flood": 2,
    "syn_flood": 3,
    "icmp_flood": 4
}
```

Después del mapeo se comprueba que no existan clases desconocidas como se observa en la siguiente Figura 82.

Figura 82

Implementación del sistema del modelo SHAPASH - comprobación de clases

```
y = (
    y_raw.astype(str)
    .str.strip()
    .str.lower()
    .map(label_map)
)

if y.isna().sum() > 0:
    invalid_labels = y_raw[y.isna()].unique()
    raise ValueError(
        f"Hay etiquetas no reconocidas en label: {invalid_labels}"
    )
```

Finalmente, las etiquetas se convierten en valores enteros y en una serie de datos. Tal cual como se evidencia en la siguiente Figura 83.

Figura 83

Implementación del sistema del modelo SHAPASH - conversión de las etiquetas en valores enteros y serie de datos

```
y = y.astype(int)
y_series = pd.Series(y, name="target")
```

3.4.5.10 Escalamiento de las características. Las variables de entrada son transformadas utilizando el escalador cargado previamente. Como se observa en la siguiente Figura 84.

Figura 84

Implementación del sistema del modelo SHAPASH - escalamiento de las características

```
X_scaled = scaler.transform(X.values)
```

En la siguiente Figura 85 se puede observar que, una vez realizadas las transformaciones por medio del escalador, los datos escalados se convierten nuevamente en un dataframe.

Figura 85

Implementación del sistema del modelo SHAPASH - conversión de los datos escalados a dataframe

```
X_scaled_df = pd.DataFrame(
    X_scaled,
    columns=feature_names
)
```

3.4.5.11 Definición del diccionario de clases. Se establece un diccionario para relacionar los valores numéricos con los nombres de las clases que se mostrarán en el dashboard.

Figura 86

Implementación del sistema del modelo SHAPASH - definición de diccionario de clases

```
label_dict = {
    0: "normal",
    1: "icmp_flood",
    2: "http_flood",
    3: "syn_flood",
    4: "udp_flood"
}
```

3.4.5.12 Creación del explicador SHAPASH. Se crea una instancia de SmartExplainer vinculando el modelo, los nombres de las características y las clases.

Figura 87

Implementación del sistema del modelo SHAPASH - creación del explicador

```
xpl = SmartExplainer(
    model=model,
    features_dict={
        col: col for col in feature_names
    },
    label_dict=label_dict
)
```

3.4.5.13 Compilación del explicador. El explicador se compila utilizando las características escaladas y las etiquetas objetivo. Esto se lo evidencia en la siguiente Figura 88.

Figura 88

Implementación del sistema del modelo SHAPASH - compilación del explicador

```
xpl.compile(
    x=X_scaled_df,
    y_target=y_series
)
```

En esta etapa se calculan las contribuciones de las variables utilizadas para interpretar las predicciones. Una vez compilado el explicador se generan las visualizaciones necesarias para analizar el comportamiento del modelo.

Primero, se presenta la importancia global de las características, permitiendo identificar cuáles variables tienen mayor influencia en las predicciones.

Figura 89

Compilación del explicador - importación global de las características

```
xpl.plot.features_importance()
```

Seguido en que la matriz de confusión para comparar las clases reales con las clases predichas.

Figura 90

Compilación del explicador - comparación de la matriz de confusión entre las clases reales con las clases predichas

```
xpl.plot.confusion_matrix_plot()
```

A continuación, la explicación local de una observación específica, indicando cómo contribuye cada característica al resultado obtenido.

Figura 91

Compilación del explicador - explicación de cada característica con respecto al resultado obtenido

```
xpl.plot.local_plot(index=0)
```

Y, por último, la relación entre una característica seleccionada y su contribución dentro de las predicciones realizadas por el modelo.

Figura 92

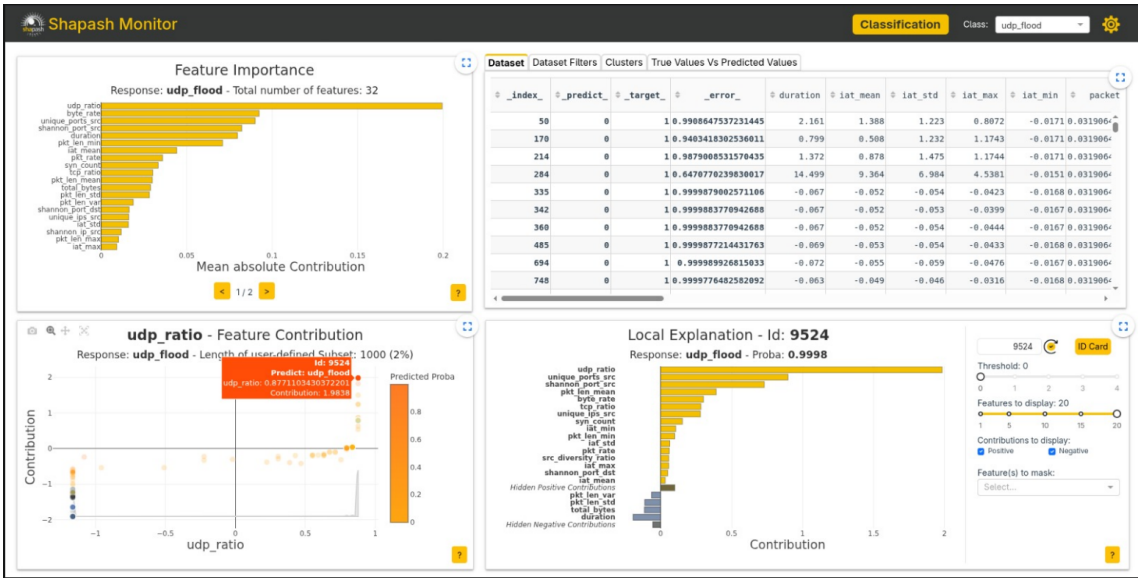
Compilación del explicador - relación entre las características con respecto a su contribución en las predicciones realizadas por el modelo

```
xpl.plot.contribution_plot(
    col=feature_names[0]
)
```

Una vez realizado todas las visualizaciones, se puede realizar la ejecución de la aplicación web interactiva de SHAPASH a través de su comando xpl.run_app (). Todo esto con la finalidad de que el dashboard permita explorar las características más importantes, revisar los registros del conjunto de datos, analizar la contribución de las variables y consultar explicaciones locales de cada predicción, tal como se lo puede llegar a observar en la siguiente Figura 93.

Figura 93

Dashboard final del modelo interpretativo SHAPASH



Capítulo IV: Evaluación de Resultados

En este capítulo se presentan y analizan los resultados obtenidos mediante la ejecución del plan de ataques en tres diferentes escenarios, red sin sistema de seguridad, red con el sistema de seguridad activo y sistema de seguridad externo. Las pruebas contemplan tráfico ICMP, TCP y UDP en condiciones normales, maliciosas y mixtas, evaluando parámetros como latencia, pérdida de paquetes, jitter, consumo de recursos, disponibilidad de los servicios, alertas e IP bloqueadas.

Los resultados obtenidos se analizan posteriormente, bajo la metodología STRIDE, poniendo el foco en la categoría de servicio, dado que, los ataques ejecutados tienen el objetivo de perjudicar la disponibilidad de la red y saturar el controlador SDN. La comparación de los distintos escenarios permite obtener la efectividad del sistema para detectar, mitigar y reducir el impacto de ataques DDoS de bajo caudal sin perjudicar el tráfico legítimo.

4.1 Plan de ataques

El plan de ataques establece una serie controlada de pruebas para evaluar el comportamiento de la red ante el tráfico normal, malicioso y mixto, la cual considera tres escenarios de prueba: una red sin sistema de seguridad, estableciendo una línea base; la red con el sistema de seguridad integrado, donde se revisará la detección, la clasificación y la respuesta ante un ataque; y un escenario diferente, donde el sistema de seguridad se coloca fuera del camino de la red, destinado a verificar la capacidad de análisis y de detección sin interferir directamente del tráfico de la red.

Las pruebas abarcarán los protocolos ICMP, UDP y TCP, donde en el protocolo ICMP se plantearán las condiciones para el tráfico normal, ataques y tráfico mixto; en el protocolo UDP, se analizará la secuencia de una llamada VoIP normal y una llamada VoIP bajo ataque, dado que UDP

es muy sensible a la latencia y a la pérdida de paquetes, de tal forma que el análisis de un flujo de voz puede ser de gran relevancia para la evaluación de la seguridad.

4.1.1 Criterios considerados para el desarrollo del plan

Para mantener una estructura ordenada de pruebas, se establecen criterios de reconocimiento, generación de tráfico, validación de conectividad y observación del impacto en la red, como se lo puede llegar a observar en la siguiente Tabla 21. Estos criterios permiten que cada caso de prueba tenga una finalidad definida y que sus resultados puedan compararse entre escenarios.

Tabla 23

Criterios considerados para el desarrollo del plan de ataques

Criterio	Descripción
TR-1	Identificar el segmento de red, los nodos normales, los nodos atacantes y el objetivo de la prueba.
TR-2	Reconocer el tipo de tráfico que será generado durante la prueba, ya sea ICMP, UDP o TCP.
TR-3	Verificar la conectividad entre origen, destino y servicios antes de iniciar cada caso de prueba.
TR-4	Registrar el comportamiento del tráfico durante la prueba mediante capturas, métricas del controlador, registros del dashboard y estado de los flujos.
ID-1	Ejecutar pruebas con hosts legítimos para establecer el comportamiento normal de la red.

Criterio	Descripción
ID-2	Ejecutar pruebas con hosts atacantes para analizar el comportamiento de la red ante tráfico malicioso.
ID-3	Ejecutar pruebas mixtas donde coexistan usuarios legítimos y atacantes dentro del mismo intervalo experimental.
ID-4	Comparar el comportamiento de la red entre los escenarios sin seguridad, con seguridad y con sistema de seguridad externo.

4.1.2 Nomenclatura de los casos de prueba

Con el objetivo de identificar de forma clara cada caso, se establece la nomenclatura PA-ES#-PROTO-##, donde PA corresponde a Plan de Ataques, ES# representa el escenario evaluado, PROTO identifica el protocolo de la prueba y ## corresponde al número secuencial del caso. Tal como se lo evidencia en el Tabla 24.

Tabla 24

Nomenclatura de los casos de prueba

Elemento	Significado
PA	Plan de ataques
ES1	Primer escenario: sin sistema de seguridad
ES2	Segundo escenario: con sistema de seguridad
ES3	Tercer escenario: sistema de seguridad externo o fuera del trayecto principal
ICMP	Pruebas basadas en tráfico ICMP

Elemento	Significado
UDP	Pruebas basadas en tráfico UDP
TCP	Pruebas basadas en tráfico TCP
01, 02, 03…	Número secuencial del caso de prueba

4.1.3 Plan general de ataques por escenarios

Tabla 25

Plan general de ataques por escenarios

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
PA-ES1-ICMP-01	Sin sistema de seguridad	ICMP	Solo normales	TR-1, TR-2, TR-3, ID-1	Se generan solicitudes ICMP desde hosts legítimos hacia los destinos definidos en la topología.	Establecer una línea base de conectividad, latencia y pérdida de paquetes sin presencia de tráfico malicioso.
PA-ES1-ICMP-02	Sin sistema de seguridad	ICMP	Solo atacantes	TR-1, TR-2, TR-4, ID-2	Los nodos atacantes generan tráfico ICMP malicioso hacia los objetivos definidos.	Evaluar el impacto del tráfico ICMP malicioso sobre el controlador, los switches y la disponibilidad de la red sin mecanismos de defensa.

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
PA-ES1-ICMP-03	Sin sistema de seguridad	ICMP	Mixto	TR-1, TR-2, TR-3, TR-4, ID-3	Se ejecuta tráfico ICMP legítimo y malicioso de forma simultánea.	Analizar cómo el tráfico de ataque afecta la conectividad de usuarios legítimos cuando no existe sistema de seguridad.
PA-ES1-UDP-04	Sin sistema de seguridad	UDP	Llamada normal sin ataque	TR-1, TR-2, TR-3, ID-1	Se establece una llamada normal entre hosts o servicios VoIP sin tráfico malicioso.	Obtener valores base de retardo, variación de retardo, pérdida de paquetes y estabilidad del servicio UDP/VoIP.
PA-ES1-UDP-05	Sin sistema de seguridad	UDP	Llamada con ataque	TR-1, TR-2, TR-4, ID-3	Se mantiene una llamada normal mientras se genera tráfico UDP malicioso desde los atacantes.	Determinar el deterioro del servicio VoIP ante tráfico UDP malicioso sin protección activa.

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
PA-ES1-TCP-06	Sin sistema de seguridad	TCP	Solo normales	TR-1, TR-2, TR-3, ID-1	Los hosts legítimos acceden a servicios TCP de forma normal.	Establecer una línea base del rendimiento TCP en términos de conexión, disponibilidad y transferencia.
PA-ES1-TCP-07	Sin sistema de seguridad	TCP	Solo atacantes	TR-1, TR-2, TR-4, ID-2	Los atacantes generan tráfico TCP malicioso hacia los servicios u objetivos definidos.	Evaluar el impacto del tráfico TCP malicioso sobre la disponibilidad de servicios y el procesamiento de flujos.
PA-ES1-TCP-08	Sin sistema de seguridad	TCP	Mixto	TR-1, TR-2, TR-3, TR-4, ID-3	Se combinan conexiones TCP legítimas con tráfico TCP malicioso.	Medir la degradación de los servicios TCP legítimos en ausencia de un sistema de seguridad.

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
PA-ES2-ICMP-01	Con sistema de seguridad	ICMP	Solo normales	TR-1, TR-2, TR-3, ID-1	Se genera tráfico ICMP legítimo con el sistema de seguridad activo.	Verificar que el sistema no bloquee tráfico normal y que no produzca falsos positivos relevantes.
PA-ES2-ICMP-02	Con sistema de seguridad	ICMP	Solo atacantes	TR-1, TR-2, TR-4, ID-2	Se genera tráfico ICMP malicioso con el sistema de seguridad activo.	Evaluar la capacidad de detección, alerta y mitigación frente a tráfico ICMP malicioso.
PA-ES2-ICMP-03	Con sistema de seguridad	ICMP	Mixto	TR-1, TR-2, TR-3, TR-4, ID-3	Coexisten solicitudes ICMP normales y tráfico ICMP malicioso.	Verificar si el sistema diferencia tráfico legítimo y malicioso, manteniendo la conectividad de usuarios normales.
PA-ES2-UDP-04	Con sistema de seguridad	UDP	Llamada normal sin ataque	TR-1, TR-2, TR-3, ID-1	Se realiza una llamada normal con el sistema de seguridad activo.	Comprobar que el sistema de seguridad no afecte la calidad

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
						del servicio VoIP en condiciones normales.
PA-ES2-UDP-05	Con sistema de seguridad	UDP	Llamada con ataque	TR-1, TR-2, TR-4, ID-3	Se mantiene una llamada mientras se genera tráfico UDP malicioso.	Evaluar si el sistema de seguridad reduce el impacto del ataque sobre la llamada y mantiene la disponibilidad del servicio.
PA-ES2-TCP-06	Con sistema de seguridad	TCP	Solo normales	TR-1, TR-2, TR-3, ID-1	Los hosts legítimos acceden a servicios TCP con el sistema de seguridad activo.	Validar que el sistema permita conexiones legítimas sin degradación significativa.
PA-ES2-TCP-07	Con sistema de seguridad	TCP	Solo atacantes	TR-1, TR-2, TR-4, ID-2	Los atacantes generan tráfico TCP malicioso	Medir la capacidad del sistema para detectar, registrar y mitigar tráfico TCP anómalo.

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
					contra los objetivos definidos.	
PA-ES2-TCP-08	Con sistema de seguridad	TCP	Mixto	TR-1, TR-2, TR-3, TR-4, ID-3	Se combinan conexiones TCP legítimas y tráfico TCP malicioso.	Comprobar la continuidad de los servicios TCP legítimos bajo ataque con el sistema de seguridad activo.
PA-ES3-ICMP-01	Sistema de seguridad externo	ICMP	Solo normales	TR-1, TR-2, TR-3, ID-1	Se genera tráfico ICMP legítimo mientras el sistema externo observa el comportamiento de la red.	Evaluar la capacidad de monitoreo del sistema externo en condiciones normales.
PA-ES3-ICMP-02	Sistema de seguridad externo	ICMP	Solo atacantes	TR-1, TR-2, TR-4, ID-2	Se genera tráfico ICMP malicioso mientras el sistema externo registra el evento.	Determinar si el sistema externo identifica patrones ICMP maliciosos sin intervenir directamente en el tráfico.

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
PA-ES3-ICMP-03	Sistema de seguridad externo	ICMP	Mixto	TR-1, TR-2, TR-3, TR-4, ID-3	Se ejecuta tráfico ICMP normal y malicioso con monitoreo externo.	Comparar la visibilidad del sistema externo frente a condiciones mixtas de tráfico.
PA-ES3-UDP-04	Sistema de seguridad externo	UDP	Llamada normal sin ataque	TR-1, TR-2, TR-3, ID-1	Se establece una llamada UDP normal mientras el sistema externo registra el tráfico.	Obtener evidencia del comportamiento de la llamada sin intervención del sistema de seguridad.
PA-ES3-UDP-05	Sistema de seguridad externo	UDP	Llamada con ataque	TR-1, TR-2, TR-4, ID-3	Se mantiene una llamada UDP mientras existe tráfico malicioso observado por el sistema externo.	Evaluar la capacidad de observación del sistema externo y comparar el deterioro del servicio frente al escenario con seguridad activa.

Caso de prueba	Escenario	Protocolo	Condición de tráfico	Requisitos	Resumen	Objetivo de la prueba
PA-ES3-TCP-06	Sistema de seguridad externo	TCP	Solo normales	TR-1, TR-2, TR-3, ID-1	Se generan conexiones TCP legítimas con monitoreo externo.	Registrar el comportamiento normal de los servicios TCP desde un sistema de seguridad no intrusivo.
PA-ES3-TCP-07	Sistema de seguridad externo	TCP	Solo atacantes	TR-1, TR-2, TR-4, ID-2	Se genera tráfico TCP malicioso mientras el sistema externo observa el evento.	Evaluar la visibilidad del sistema externo frente a patrones TCP inusuales.
PA-ES3-TCP-08	Sistema de seguridad externo	TCP	Mixto	TR-1, TR-2, TR-3, TR-4, ID-3	Se combinan conexiones TCP legítimas y tráfico TCP malicioso bajo observación externa.	Comparar la capacidad de análisis del sistema externo frente al escenario con mitigación activa.

4.1.4 Descripción operativa de los escenarios

Durante la ejecución de cada caso del plan de ataques se deben registrar parámetros técnicos que permitan comparar los tres escenarios, como se lo puede observar en la siguiente Tabla 26. Estos parámetros permiten validar el impacto del ataque y la efectividad del sistema de seguridad.

Tabla 26

Descripción operativa de los escenarios

Parámetro	Descripción	Aplicación dentro del plan
Latencia	Tiempo de respuesta entre origen y destino.	Comparar tráfico normal y tráfico bajo ataque.
Pérdida de paquetes	Porcentaje de paquetes no entregados correctamente.	Evaluar degradación de ICMP, UDP y TCP.
Jitter	Variación del retardo entre paquetes.	Especialmente importante en llamadas UDP/VoIP.
Ancho de banda consumido	Cantidad de tráfico generado durante cada prueba.	Identificar saturación de enlaces o servicios.
Uso de CPU del controlador	Consumo de procesamiento del controlador SDN.	Determinar si el ataque afecta al plano de control.
Uso de memoria del controlador	Consumo de RAM durante los eventos de tráfico.	Evaluar estabilidad del controlador.
Cantidad de flujos instalados	Número de reglas o flujos generados en los switches.	Analizar impacto en el plano de datos.

Parámetro	Descripción	Aplicación dentro del plan
Alertas generadas	Eventos detectados por el sistema de seguridad.	Validar capacidad de detección.
IPs bloqueadas o controladas	Orígenes identificados como maliciosos.	Evaluar capacidad de mitigación.
Disponibilidad del servicio	Estado funcional de VoIP, TCP, web, FTP u otros servicios.	Comparar continuidad del servicio entre escenarios.

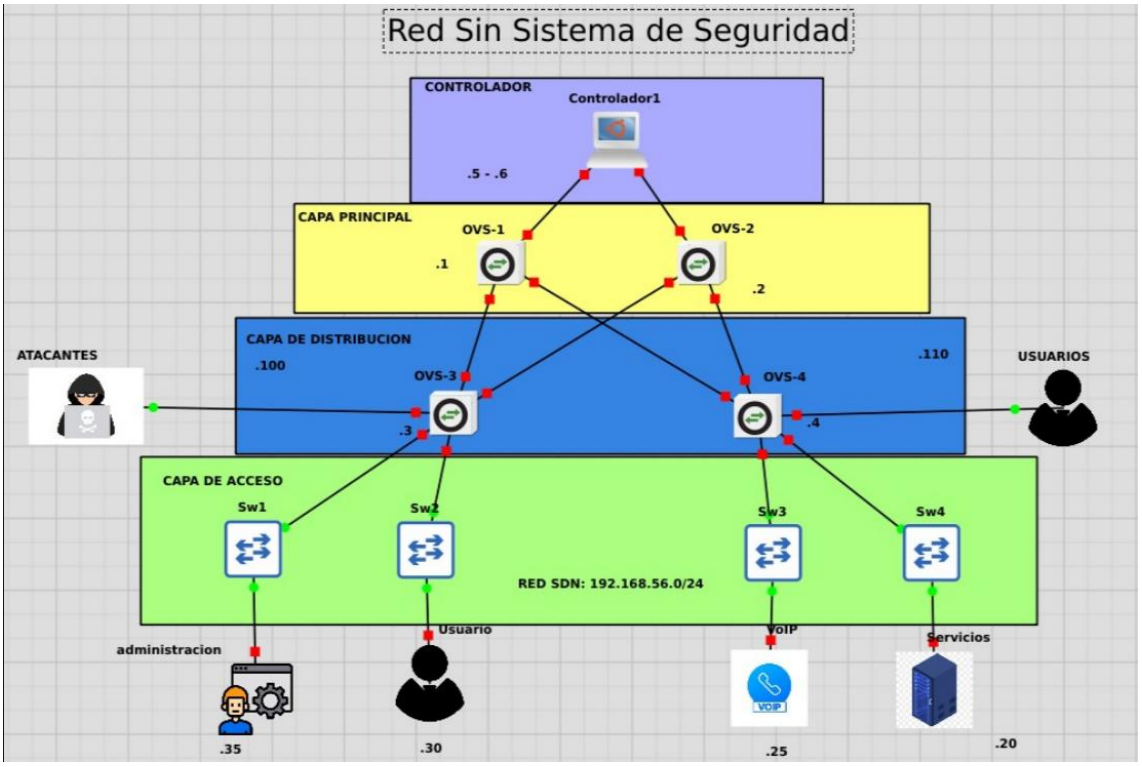
El plan de ataques propuesto organiza la evaluación de la infraestructura de una manera ordenada bajo tres condiciones experimentales: sin sistema de seguridad, con sistema de seguridad activo y con sistema de seguridad externo. Esta división facilita la comparación del comportamiento de la red ante tráfico ICMP, UDP y TCP, considerando casos normales, maliciosos y mixtos. De esta manera, el plan no solo permite identificar el impacto de los ataques, sino también validar la efectividad del mecanismo de seguridad implementado y su capacidad para conservar la disponibilidad de los servicios legítimos dentro del entorno experimenta.

4.1.4.1 Primer escenario: red sin sistema de seguridad. La red SDN fue evaluada sin contar con mecanismos de detección y de mitigación, por lo que el controlador Ryu solo gestionaba switches y el reenvío de paquetes. Su objetivo fue el de establecer una línea de base mediante el empleo de pruebas de tráfico normal, malicioso y mixto sobre los protocolos ICMP, TCP y UDP. En la evaluación de la red se analizaron indicadores como la transmisión y la pérdida de paquetes, el estado de los servicios, las retransmisiones los jitters y el consumo del CPU y memoria del controlador.

Verificación de la topología. Antes de ejecutar las pruebas se verificó que todos los componentes de la topología estuvieran levantados y correctamente interconectados. La infraestructura estuvo conformada por el controlador Ryu, switches Open vSwitch, usuarios normales, usuarios atacantes, un servidor web y un servidor de voz IP como se lo puede observar en la siguiente Figura 94.

Figura 94

Topología - Red sin sistema de seguridad



Esta comprobación inicial fue necesaria para asegurar que los resultados posteriores correspondieran al comportamiento del tráfico generado y no a errores de configuración, dispositivos apagados o fallas de conectividad.

La red experimental utilizó el segmento 192.168.56.0/24. Dentro de esta red se identificaron las siguientes direcciones principales:

- Usuarios normales: 192.168.56.110.
- Usuarios atacantes: 192.168.56.100.
- Servidor web: 192.168.56.20.
- Servidor VoIP: 192.168.56.25.
- Controlador Ryu: 192.168.56.5 y 192.168.56.6.

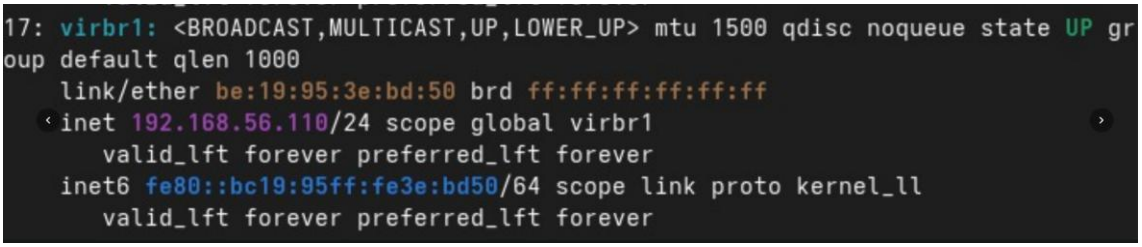
Estas direcciones se utilizaron posteriormente como orígenes o destinos de las diferentes pruebas.

Verificación del direccionamiento IP. La primera comprobación técnica consistió en revisar las interfaces y direcciones IPv4 de los dispositivos. Para ello, se ejecutaron comandos de consulta de red dentro de las máquinas o contenedores correspondientes.

Usuarios normales. En la siguiente Figura 95 correspondiente a los usuarios normales se observó que la interfaz virbr1 se encontraba en estado UP, lo cual indicó que estaba activa y disponible para transmitir tráfico. A esta interfaz se le asignó la dirección 192.168.56.110/24.

Figura 95

Verificación del direccionamiento IP – Usuarios normales

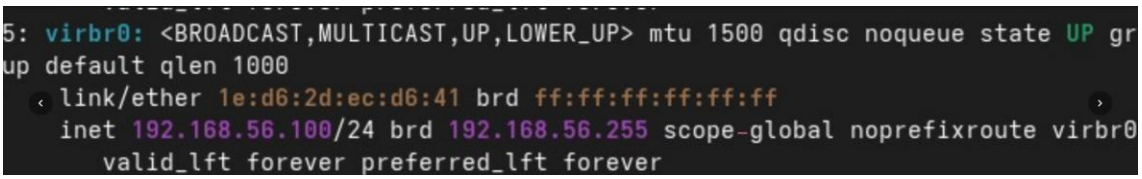


El prefijo /24 indicó que el dispositivo pertenecía al segmento 192.168.56.0 con máscara 255.255.255.0. Esta dirección fue utilizada como referencia para identificar el tráfico legítimo generado por los usuarios normales durante las pruebas.

Usuarios atacantes. En los usuarios atacantes se comprobó que la interfaz virbr0 también estaba en estado UP, dentro de este aspecto se menciona que la dirección asignada fue 192.168.56.100/24.

Figura 96

Verificación del direccionamiento IP – Usuarios atacantes



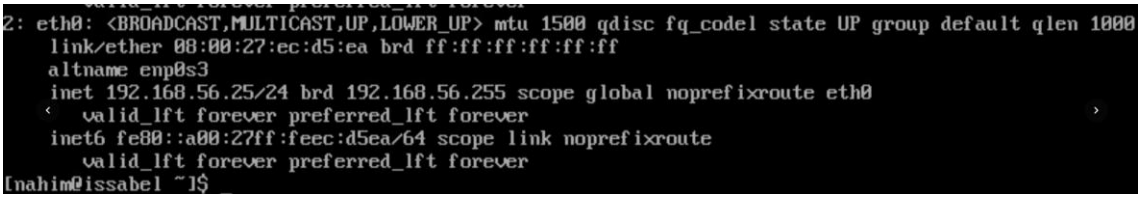
Esta dirección permitió identificar el nodo desde el cual se administraron y levantaron los bots encargados de generar los ataques. Al encontrarse dentro del mismo segmento de red los atacantes podían alcanzar al controlador y a los servidores definidos como objetivos.

- Servidor de voz IP

En el servidor VoIP se verificó que la interfaz eth0 se encontraba activa y tenía configurada la dirección 192.168.56.25/24.

Figura 97

Establecimiento del servidor VoIP

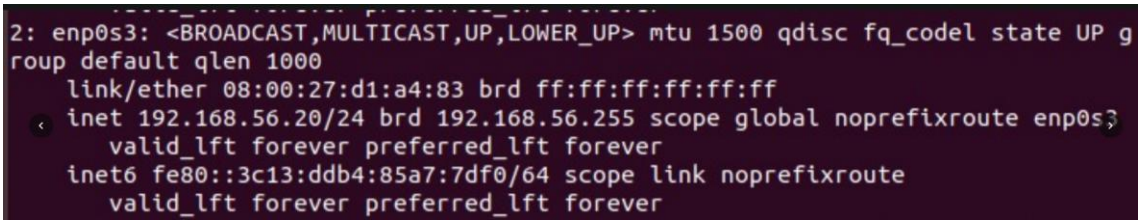


Esta comprobación fue importante debido a que los clientes de telefonía debían registrarse contra esta dirección para establecer las llamadas SIP. La misma dirección fue utilizada posteriormente como objetivo del ataque UDP Flood.

Servidor web. En el servidor web se verificó que la interfaz de red se encontraba activa y que la dirección configurada era 192.168.56.20/24

Figura 98

Establecimiento del servidor web

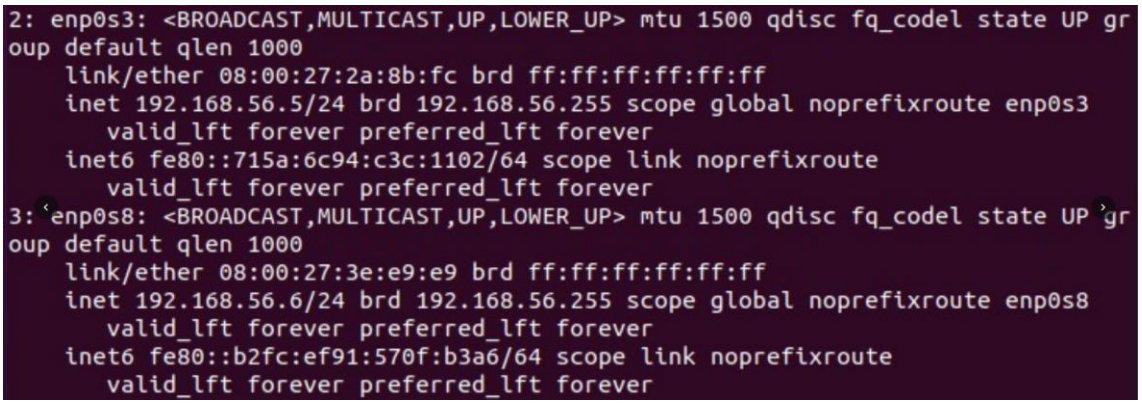


Esta dirección se utilizó como destino de las solicitudes HTTP legítimas generadas con CURL y de los ataques SYN Flood ejecutados por los bots.

Controlador Ryu. Finalmente, en el controlador se comprobaron dos interfaces configuradas dentro de la red experimental. Las direcciones observadas fueron 192.168.56.5/24 y 192.168.56.6/24.

Figura 99

Establecimiento del controlador Ryu

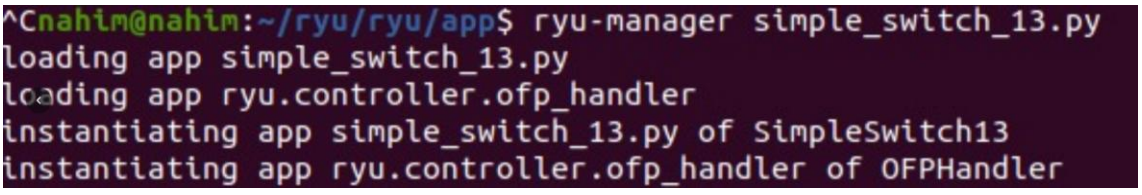


La presencia de las dos direcciones permitió que los switches Open vSwitch tuvieran configurados dos puntos de conexión hacia el plano de control. Con estas verificaciones se confirmó que los principales dispositivos se encontraban activos y dentro del mismo segmento de red.

Verificación del controlador Ryu. Después de comprobar el direccionamiento se inició el controlador Ryu mediante el comando ryu-manager simple_switch_13.py como se lo puede observar en la siguiente Figura 100.

Figura 100

Verificación del controlador Ryu



La terminal confirmó la carga de simple_switch_13.py y la inicialización de la clase SimpleSwitch13. Esta aplicación simula un switch de aprendizaje para OpenFlow 1.3. Su función consiste en recibir los primeros paquetes de los flujos, aprender las direcciones MAC de origen y determinar el puerto por el cual debe reenviarse el tráfico. En este escenario no se cargó la aplicación del sistema de seguridad, por lo que el controlador no analizó el tráfico mediante XGBoost ni ejecutó acciones de bloqueo, conduciendo a que el resultado de esta verificación confirme que el plano de control estaba operativo, pero sin mecanismos de protección frente a tráfico malicioso.

Verificación de los switches Open vSwitch. La conexión entre los switches y el controlador se comprobó mediante el comand ovs-vsctl show.

Figura 101

Verificación de los switches Open vSwitch



En la salida se observó que el bridge Open vSwitch tenía configurados los siguientes controladores:

- tcp:192.168.56.5:6653
- tcp:192.168.56.6:6653

En ambos casos se presentó el estado is_connected: true, con lo cual conlleva a mencionar que este resultado confirmó que existía comunicación activa mediante OpenFlow entre el plano de

datos y el controlador Ryu. También se observó la configuración fail_mode: standalone, la cual permite que el switch mantenga un funcionamiento básico si pierde temporalmente la comunicación con el controlador.

Además, la verificación permitió descartar problemas de conexión entre Ryu y los switches antes de comenzar las pruebas de tráfico correspondientes.

Prueba ICMP con usuarios normales.

a) Objetivo de la prueba

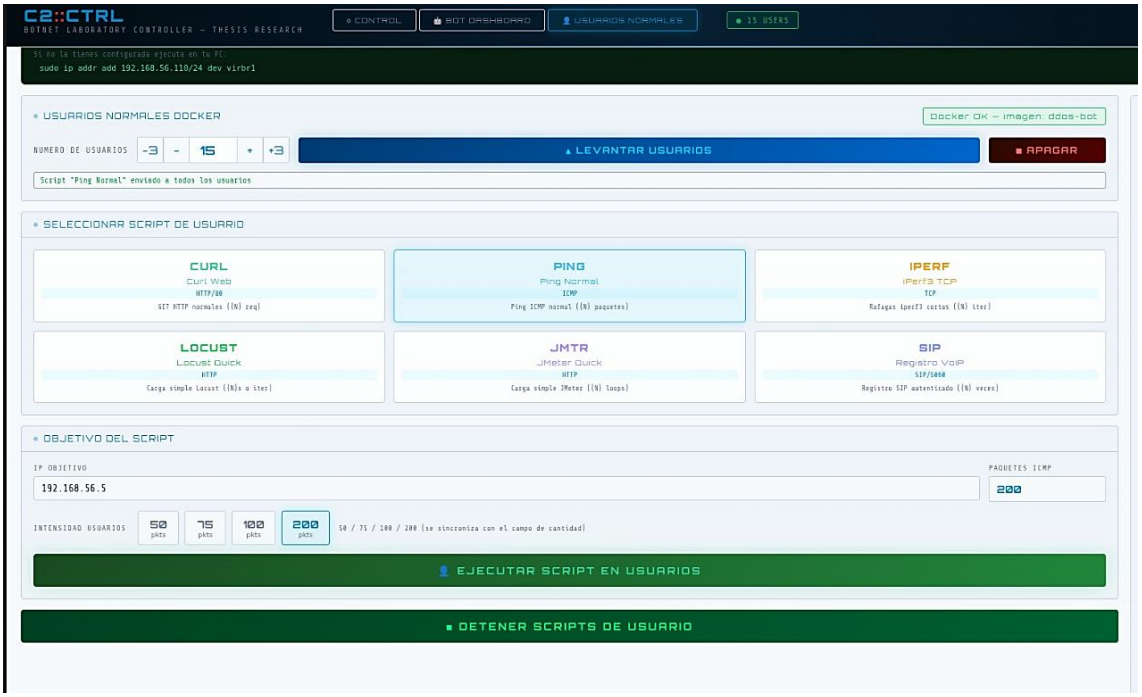
La prueba PA-ES1-ICMP-01 tuvo como finalidad establecer el comportamiento normal de la red para tráfico ICMP. Se buscó verificar la conectividad hacia el controlador, medir la pérdida de paquetes y registrar el consumo base de CPU y memoria.

b) Configuración y ejecución

Desde el dashboard se levantaron 15 usuarios normales. Posteriormente, se seleccionó el script PING y se configuró como destino la dirección 192.168.56.5 correspondiente al controlador Ryu.

Figura 102

Configuración y ejecución ICMP de los usuarios normales

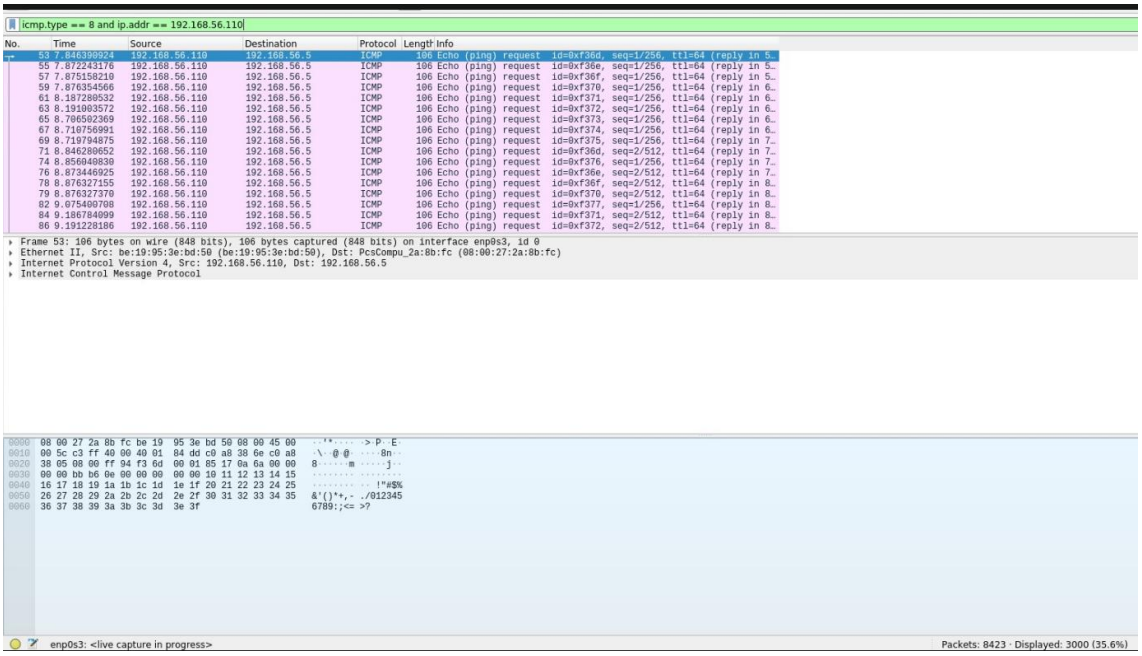


Cada usuario generó 200 solicitudes ICMP Echo Request. Por tanto, la cantidad total esperada debería ser de:

$$15\ usuarios \times 200\ solicitudes = 3000\ solicitudes\ ICMP$$

Figura 103

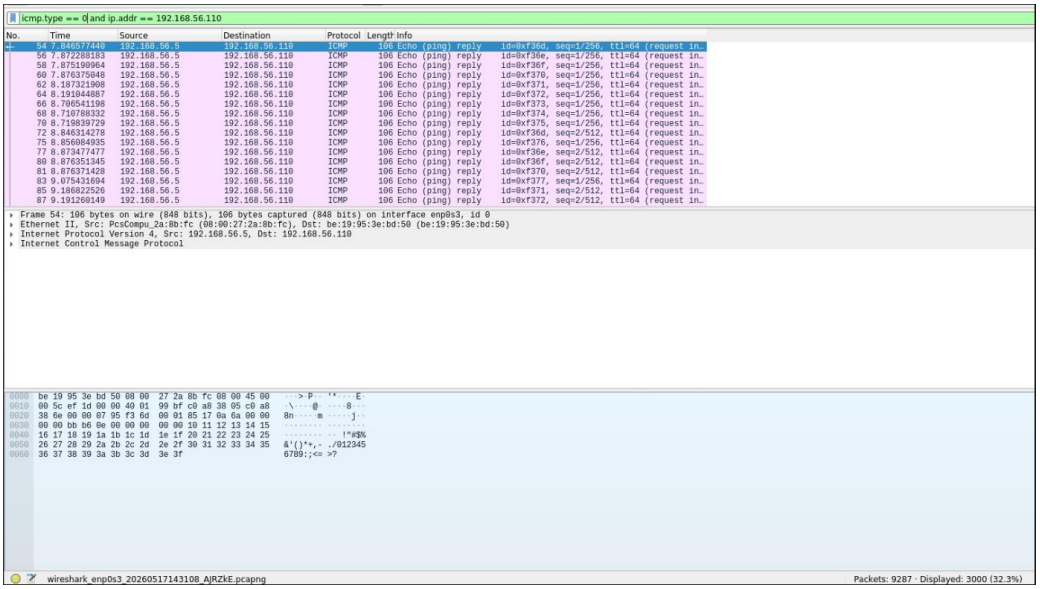
Generación de solicitudes ICMP por parte de los usuarios normales establecidos



Antes de ejecutar el tráfico se inició la captura en Wireshark. Luego se aplicaron filtros para visualizar por separado las solicitudes Echo Request y las respuestas Echo Reply.

Figura 104

Visualización de las solicitudes ICMP Echo Request y respuestas Echo Reply



En la captura de solicitudes se contabilizaron 3000 mensajes Echo Request como se lo llegó a observar en la Figura 104 dirigidos hacia el controlador. Este resultado coincidió con la cantidad configurada en el dashboard. Posteriormente, se analizaron las respuestas y se registraron 3000 mensajes Echo Reply. Esto indicó que el controlador o el destino respondió a la totalidad de las solicitudes legítimas. Llevando a establecerse una pérdida de paquetes de 0 %.

$$Pérdida = \frac{Echo\ Request\ enviados - Echo\ Reply\ recibidos}{Echo\ Request\ enviados}$$

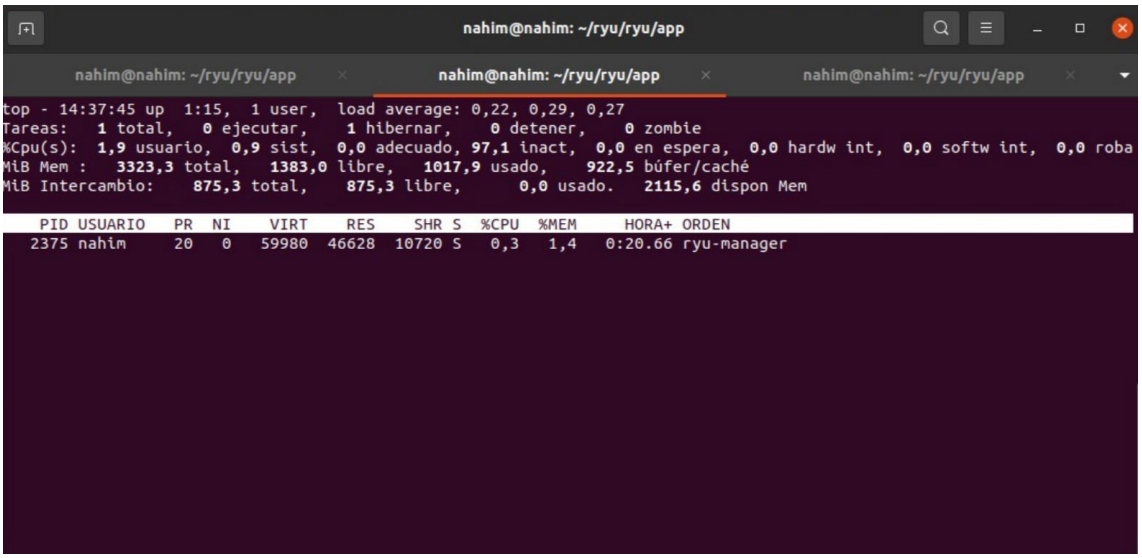
$$Pérdida\ \% = \frac{3000 - 3000}{3000} \times 100$$

$$Pérdida = 0\ \%$$

Durante la prueba el proceso ryu-manager presentó aproximadamente CPU: 0,3 % y Memoria: 1,4 %.

Figura 105

Proceso ryu-manager ICMP por parte de los usuarios normales



La ausencia de pérdida indicó que la red operó correctamente cuando únicamente existió tráfico ICMP legítimo. El bajo consumo de CPU mostró que la carga producida por los 15 usuarios

y los 3000 paquetes no representó un esfuerzo considerable para el controlador. Este resultado se utilizó como referencia para comparar las pruebas ICMP con atacantes y tráfico mixto.

Prueba ICMP únicamente con atacantes

a) Objetivo de la prueba

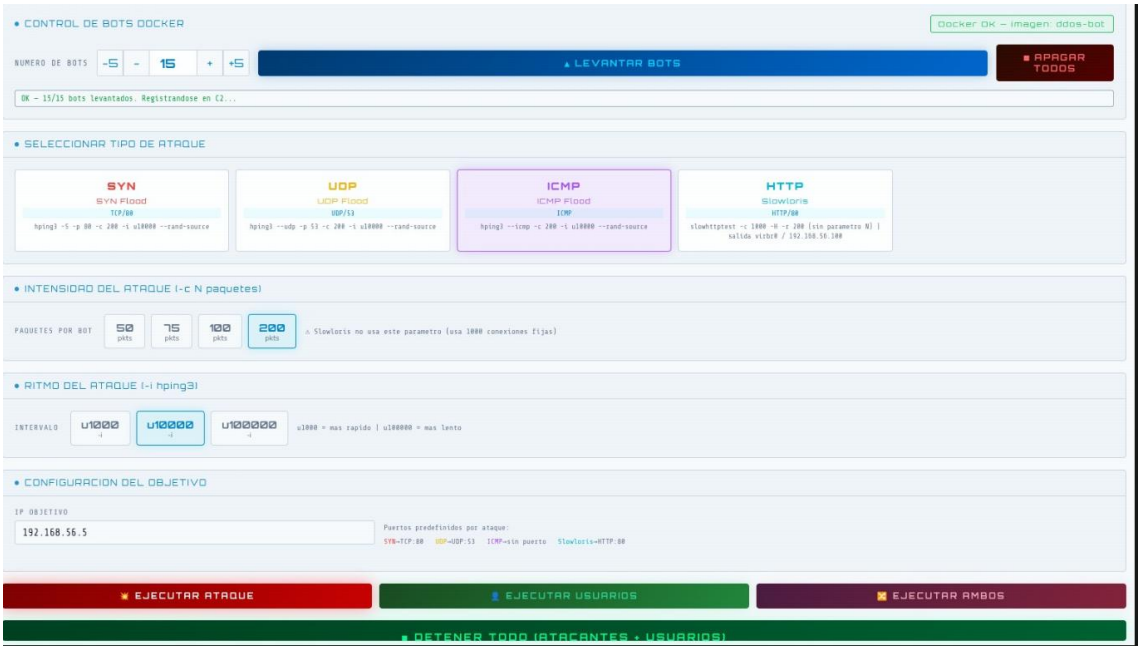
El caso PA-ES1-ICMP-02 tuvo como objetivo observar el comportamiento del controlador al recibir tráfico ICMP generado únicamente por nodos atacantes, sin disponer de reglas de detección o mitigación.

b) Configuración y ejecución

Desde el dashboard de administración se levantaron 15 bots, en los cuales se seleccionaron el ataque ICMP Flood y se estableció como objetivo la dirección 192.168.56.5.

Figura 106

Configuración y ejecución ICMP únicamente con atacantes

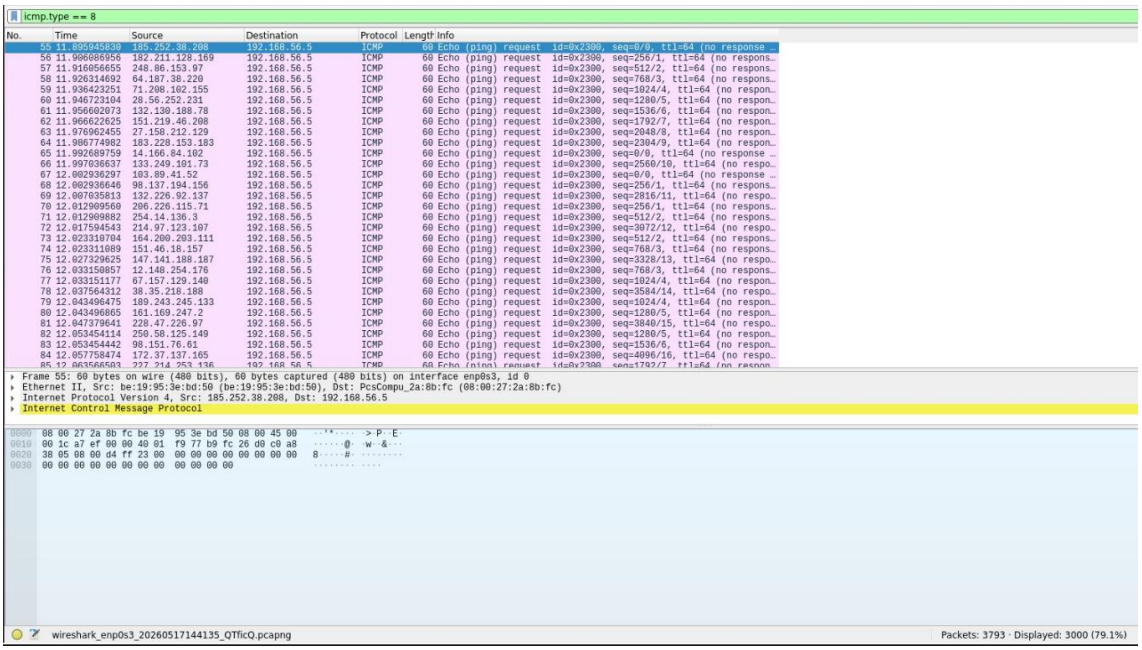


Dentro de este aspecto se puede mencionar que cada atacante envió 200 paquetes, por lo que el volumen esperado fue:

$$15\ atacantes \times 200\ solicitudes\ = 3000\ paquetes\ ICMP\ maliciosos$$

Figura 107

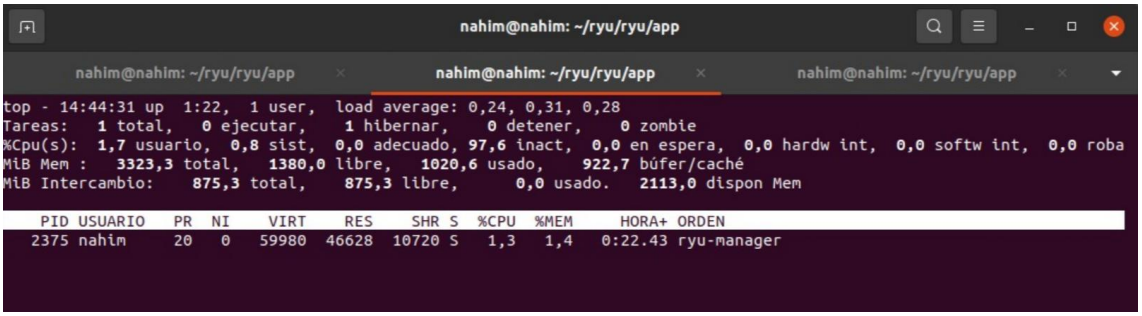
Generación de solicitudes ICMP por parte de los atacantes



Como se pudo observar en la Figura 108 el ataque fue ejecutado mientras Wireshark capturaba el tráfico dirigido hacia el controlador. En Wireshark se observaron 3000 mensajes ICMP Echo Request. En la captura se visualizaron diferentes direcciones IP de origen, lo cual se relacionó con la generación de tráfico mediante direcciones aleatorias o modificadas desde los atacantes. Durante esta prueba el proceso del controlador presentó aproximadamente un CPU: 1,3% y Memoria: 1,4%.

Figura 108

Proceso ryu-manager ICMP por parte únicamente de los atacantes



Con lo cual se pudo decir que el controlador continuó disponible y no se produjo una interrupción total del servicio, esto a pesar de que, aunque la cantidad de paquetes fue igual a la prueba normal, el consumo de CPU aumentó de 0,3 % a 1,3 %, logrando denotar que la forma de generación del tráfico malicioso produjo una carga de procesamiento superior. Todo esto debido a que el sistema de seguridad no estaba activo y por ende los paquetes de ataque fueron procesados sin ser clasificados, limitados o bloqueados. Por esta razón, el tráfico malicioso alcanzó directamente al objetivo.

Prueba ICMP mixta

a) Objetivo de la prueba

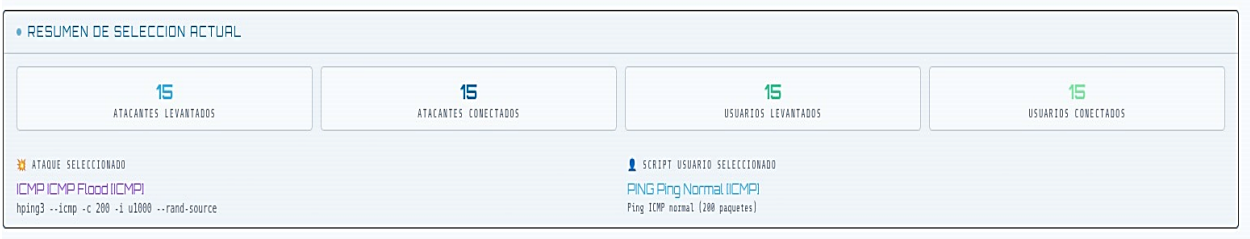
La prueba PA-ES1-ICMP-03 buscó representar una condición más cercana a una red real, donde el tráfico legítimo y malicioso circulan simultáneamente. El objetivo fue determinar si los usuarios normales mantenían la conectividad mientras el controlador recibía el ICMP Flood.

b) Configuración y ejecución

En este caso se hizo un levantamiento de manera simultánea de 15 usuarios normales scon 15 atacantes.

Figura 109

Configuración y ejecución ICMP mixta



Los usuarios normales ejecutaron PING hacia el controlador, mientras que los bots ejecutaron el ICMP Flood, logrando que cada participante genere 200 paquetes.

$$15\ usuarios\ normales \times 200\ solicitudes = 3000\ solicitudes\ ICMP$$

$$15\ atacantes \times 200\ solicitudes\ = 3000\ paquetes\ ICMP\ maliciosos$$

$$3000\ ICMP + 3000\ ICMP\ maliciosos\ = 6000\ paquetes\ totales\ esperados$$

Figura 110

Generación de solicitudes ICMP mixta

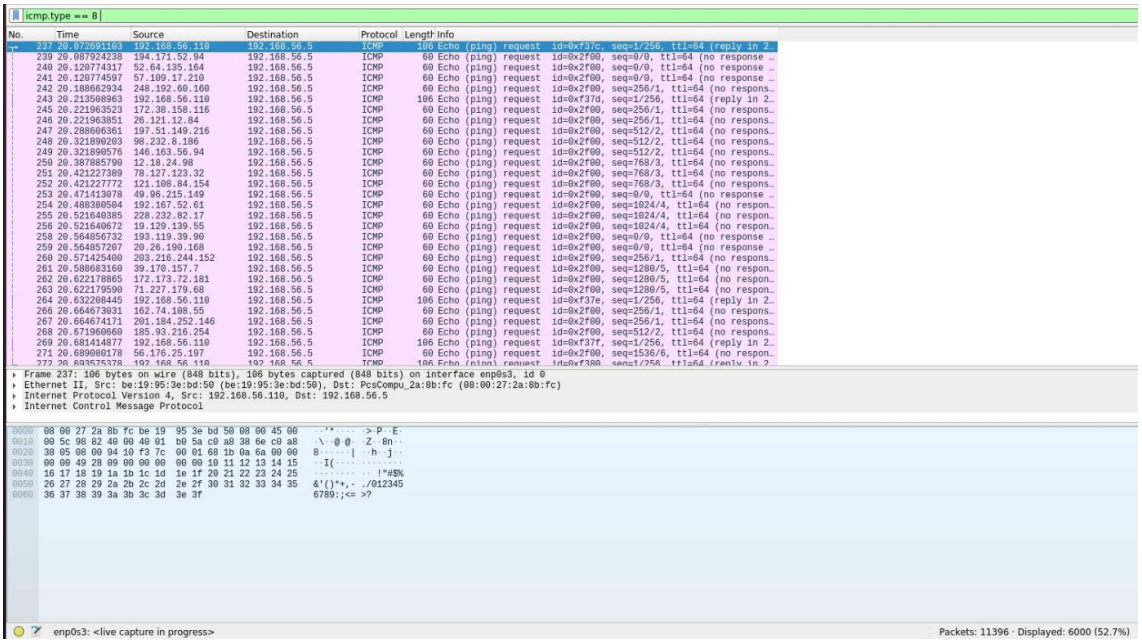
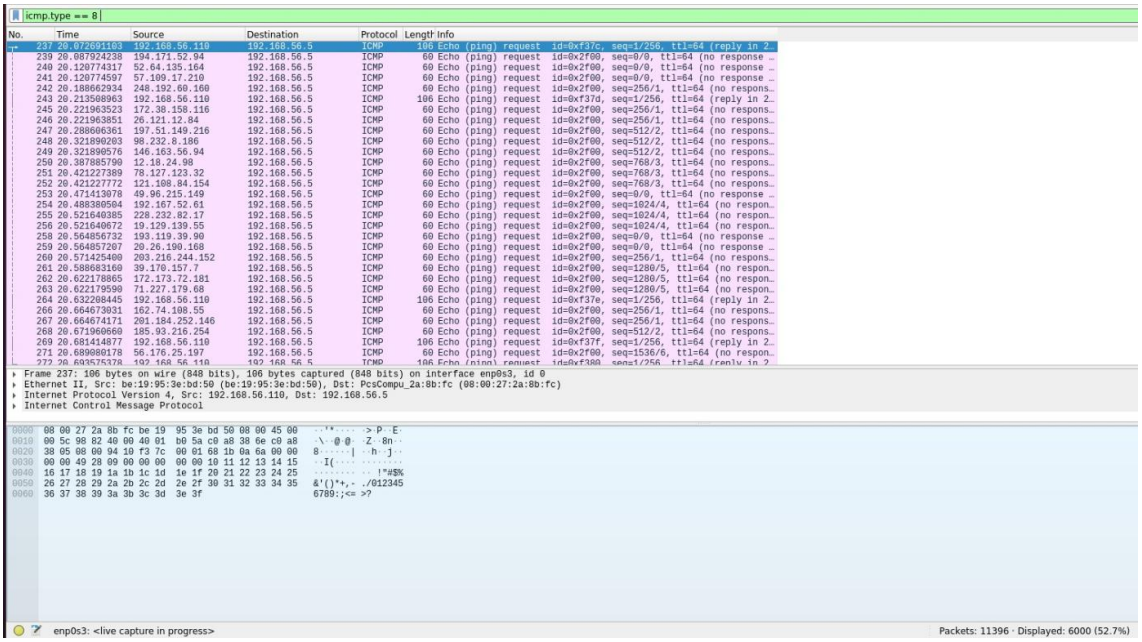


Figura 111

Generación de solicitudes ICMP observados



Como se pudo observar en las Figuras 111 en Wireshark se mostraron 9000 paquetes ICMP. Este valor no correspondió únicamente a solicitudes, sino a la suma de los siguientes mensajes.

$$Paquetes\ ICMP\ observados$$

$$= 3000\ Echo\ Request\ normales$$

$$+ 3000\ Echo\ Reply\ hacia\ los\ usuarios\ normales$$

$$+\ 3000\ Echo\ Request\ de\ ataque$$

$$Paquetes\ ICMP\ observados = 9000$$

Los usuarios normales recibieron las 3000 respuestas esperadas. Por ello, la pérdida de tráfico legítimo se mantuvo en 0 %.

$$Pérdida\ = \frac{Echo\ Request\ enviados - Echo\ Reply\ recibidos}{Echo\ Request\ enviados}$$

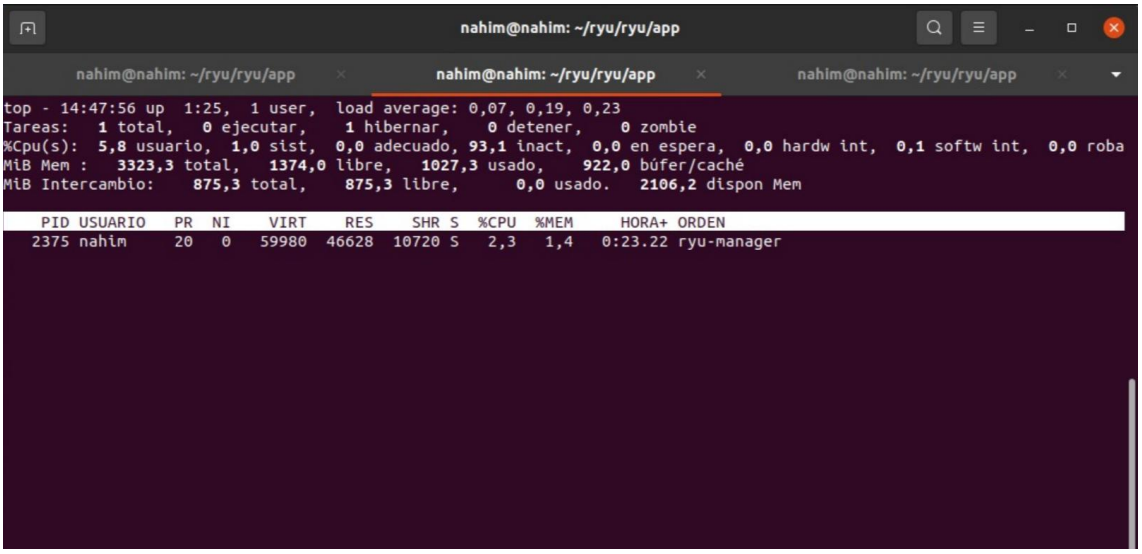
$$Pérdida\ \% = \frac{3000 - 3000}{3000} \times 100$$

$$Pérdida\ = 0\ \%$$

En la captura puntual del uso de recursos se observó aproximadamente un CPU de ryu-manager: 2,3 % y Memoria: 1,4 % como se lo puede llegar a observar en la siguiente Figura 112.

Figura 112

Proceso ryu-manager ICMP mixta



Llegando a establecer que durante el desarrollo de la prueba también se reportó un comportamiento intermitente de la comunicación.

A pesar de que no hubo pérdida en los paquetes legítimos contabilizados, la red debió procesar una cantidad de tráfico total mayor que en las pruebas individuales. Aparte de que el incremento de la CPU y la intermitencia observada señalaron que el tráfico malicioso provocó una afectación operativa, aunque no la caída completa del controlador. Este comportamiento es interesante en ataques de poca intensidad ya que la degradación puede producirse de una manera progresiva y, por tanto, no necesariamente a través de una interrupción inmediata.

Prueba TCP con usuarios normales

a) Objetivo de la prueba

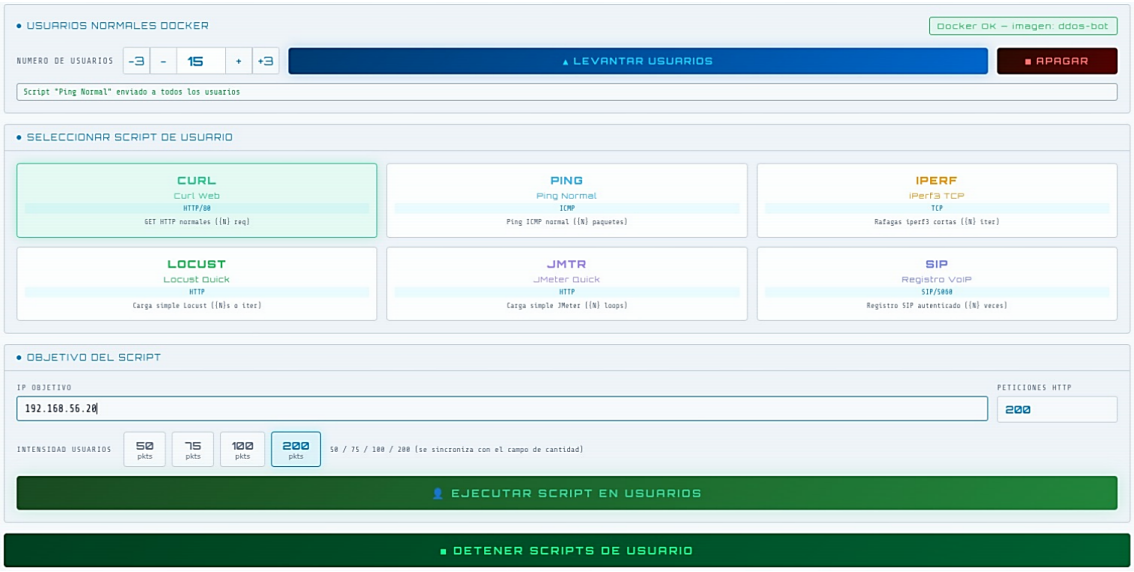
La prueba PA-ES1-TCP-06 tuvo como propósito establecer el comportamiento normal del servidor web ante solicitudes HTTP legítimas. También se buscó registrar la disponibilidad, retransmisiones, reinicios de conexión y consumo de recursos del controlador.

b) Configuración y ejecución

Desde el dashboard se levantaron 15 usuarios normales. Se seleccionó el script CURL y se configuró como objetivo la dirección del servidor web 192.168.56.20.

Figura 113

Configuración y ejecución TCP con usuarios normales



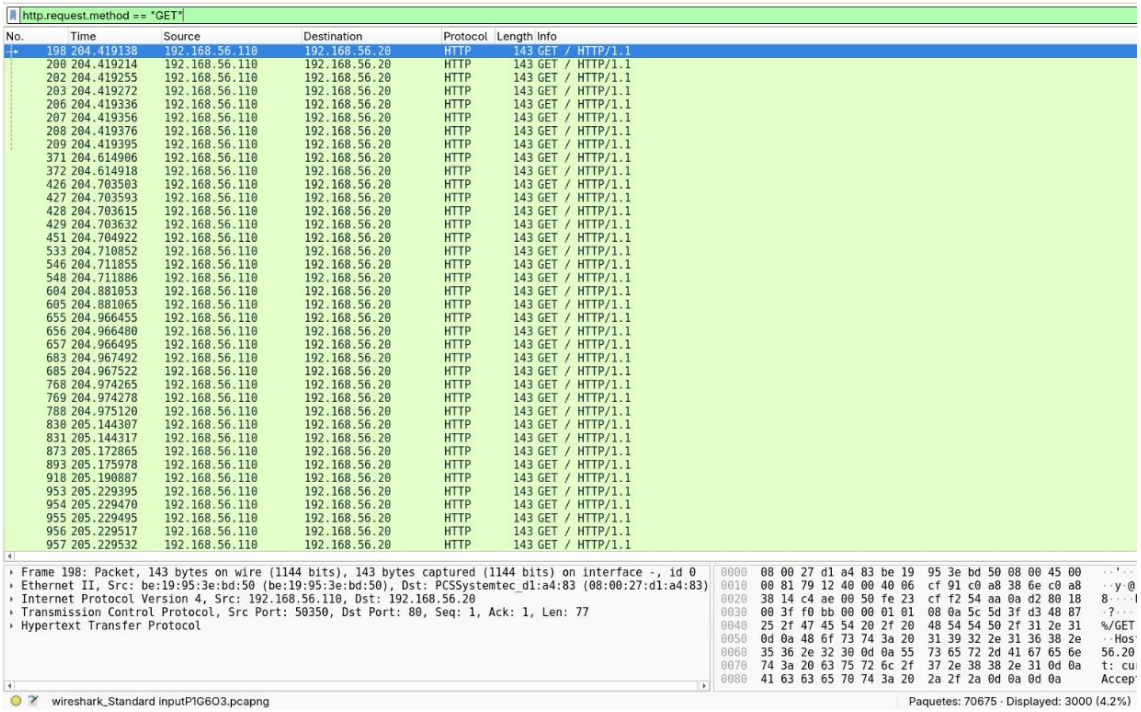
En donde cada usuario ejecutó 200 solicitudes HTTP, por lo que el total esperado fue de igual manera de 3000.

$$15\ usuarios\ normales \times 200\ solicitudes = 3000\ solicitudes\ TCP$$

Durante la ejecución se capturó el tráfico TCP y HTTP mediante Wireshark. En el cual se pudo llegar a observar que mediante el filtro correspondiente a solicitudes GET se contabilizaron 3000 peticiones HTTP dirigidas hacia el servidor web.

Figura 114

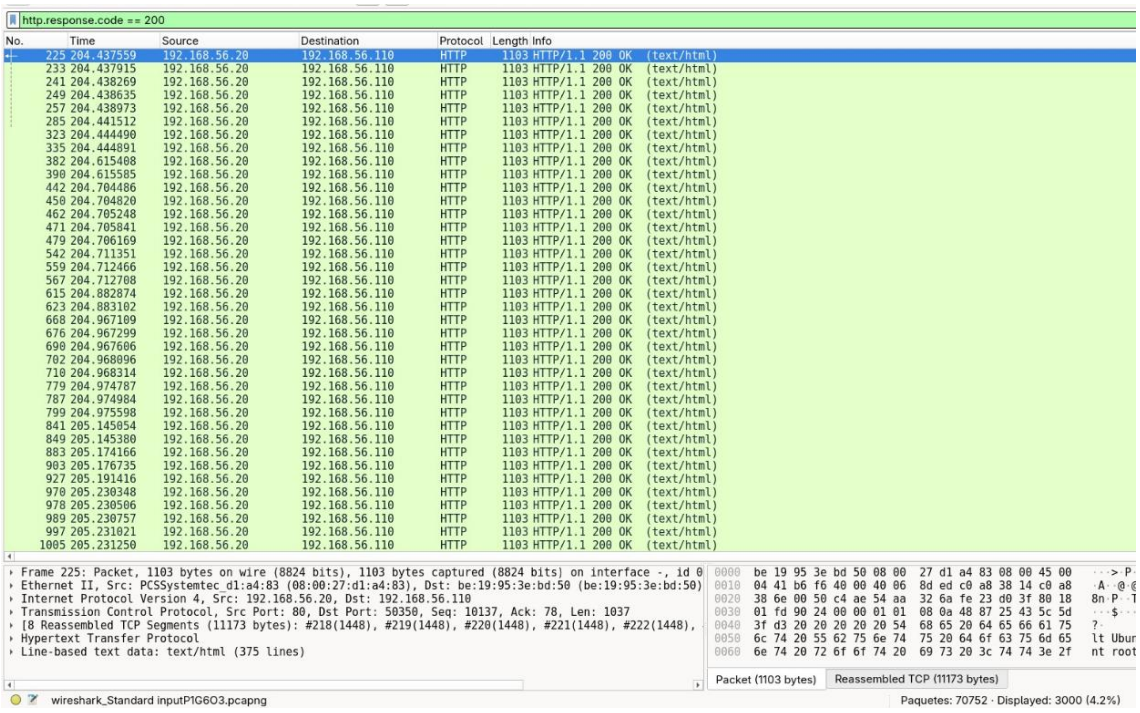
Generación de solicitudes TCP con usuarios normales



Posteriormente, se analizaron las respuestas HTTP con código 200, el cual representa que la petición fue procesada exitosamente, logrando así observar alrededor de 3000 respuestas exitosas.

Figura 115

Captura de respuestas HTTP de las solicitudes TCP con usuarios normales



Una vez localizados tanto las solicitudes GET y respuestas se pudo establecer la disponibilidad en cuanto a la continuidad del servicio mediante la siguiente ecuación.

$$Disponibilidad = \frac{Respuestas\ existosas}{Solicitudes\ enviadas}$$

$$Disponibilidad = \frac{3000}{3000} \times 100$$

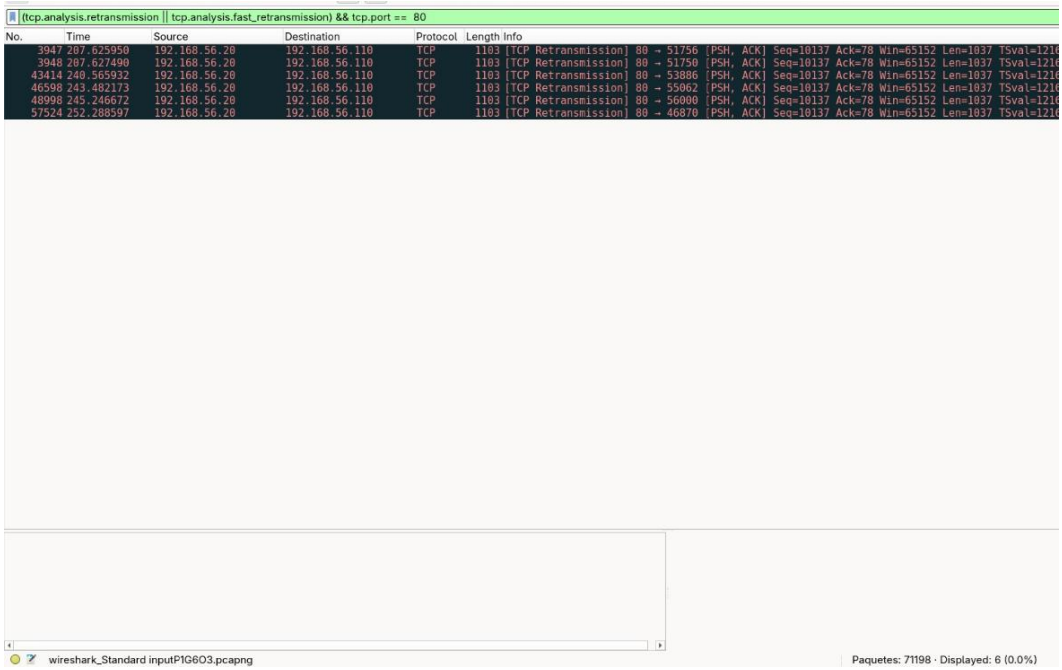
$$Disponibilidad = 100\%$$

Logrando observar que al obtener un 100% se puede mencionar que estuvo correctamente sin problemas. Además de llegar a establecer ciertas particularidades como:

- Retransmisiones TCP: 6.

Figura 116

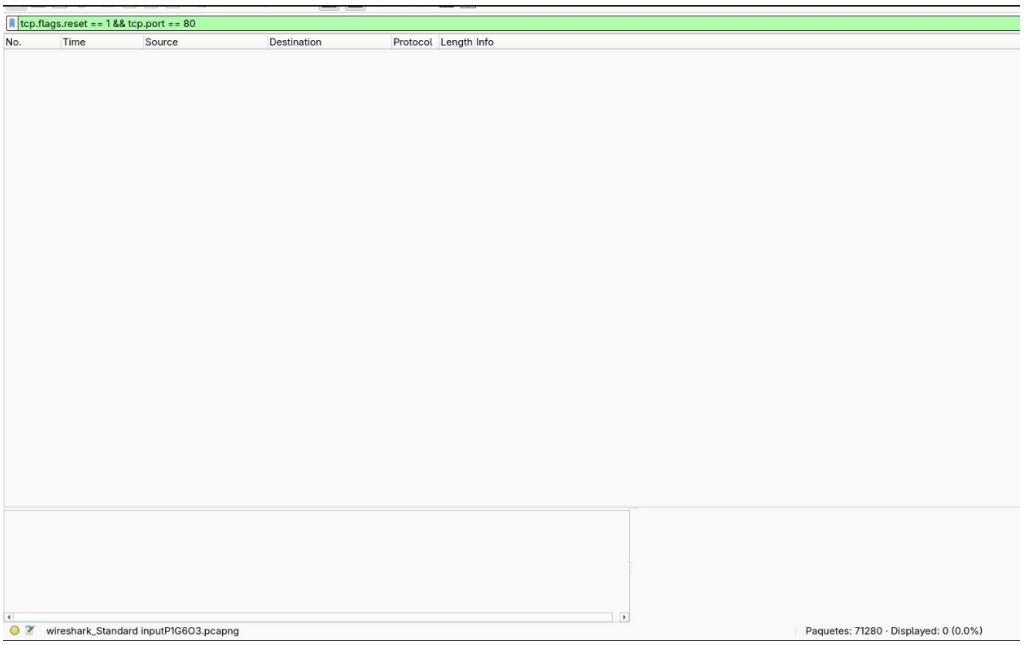
Retransmisiones TCP con usuarios normales



- Paquetes RST: 0.

Figura 117

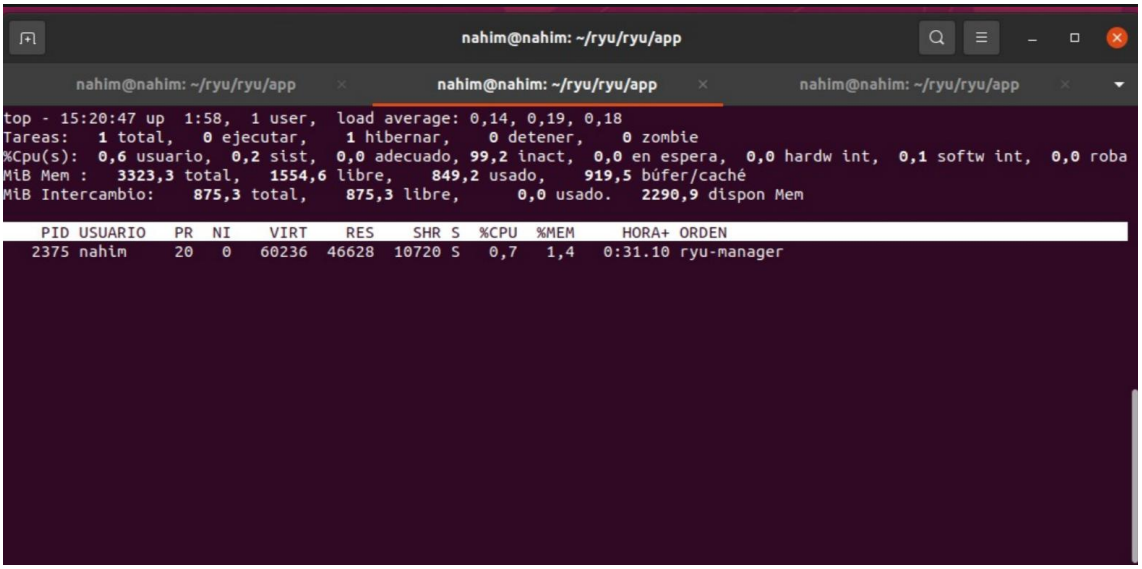
Paquetes TCP con usuarios normales



- CPU: 0,7 % y Memoria: 1,4 %.

Figura 118

Proceso ryu-manager TCP con usuarios normales



Con la detección de estos aspectos se puede llegar a mencionar que el servidor web atendió correctamente todas las solicitudes legítimas. La ausencia de paquetes RST indicó que no se produjeron cierres abruptos o reinicios forzados de las conexiones.

Las seis retransmisiones mostraron que algunos segmentos TCP tuvieron que enviarse nuevamente, aunque esto no impidió que las solicitudes fueran respondidas. Por ello, el servicio se consideró disponible y estable durante la prueba normal.

Prueba TCP con atacantes

a) Objetivo de la prueba

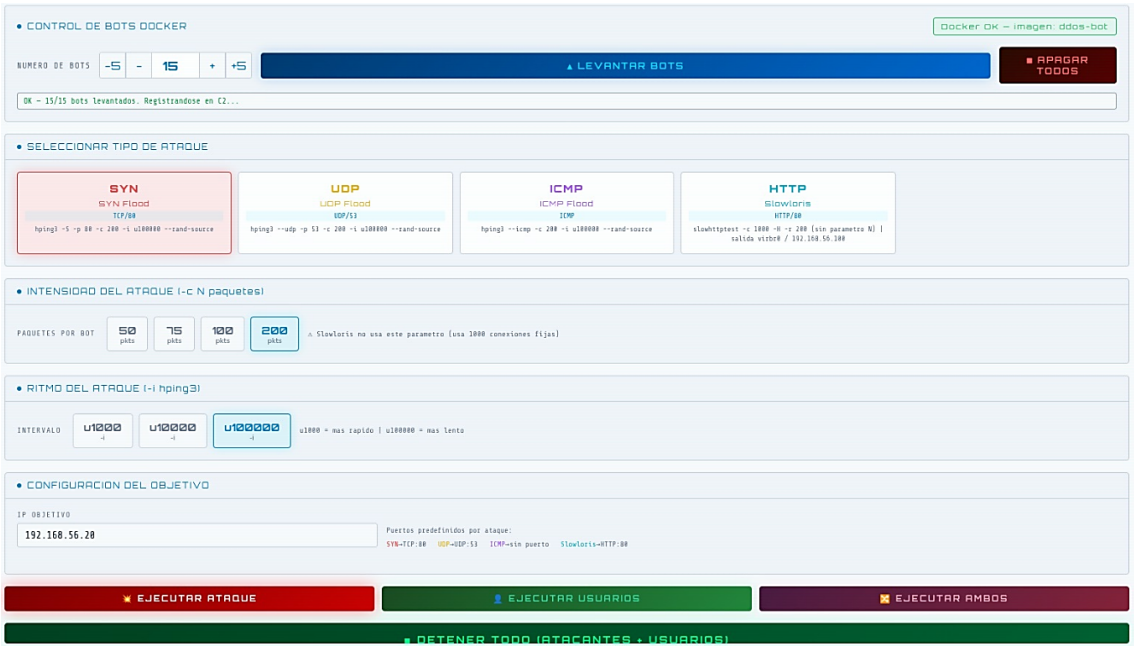
La prueba con atacantes tuvo como finalidad observar la respuesta del servidor y del controlador frente a un SYN Flood. Este tipo de ataque genera múltiples solicitudes de inicio de conexión TCP sin completar adecuadamente el establecimiento de la sesión.

b) Configuración y ejecución

Se levantaron 15 bots y se seleccionó el ataque SYN Flood con objetivo hacia el servidor web 192.168.56.20.

Figura 119

Configuración y ejecución TCP con atacantes



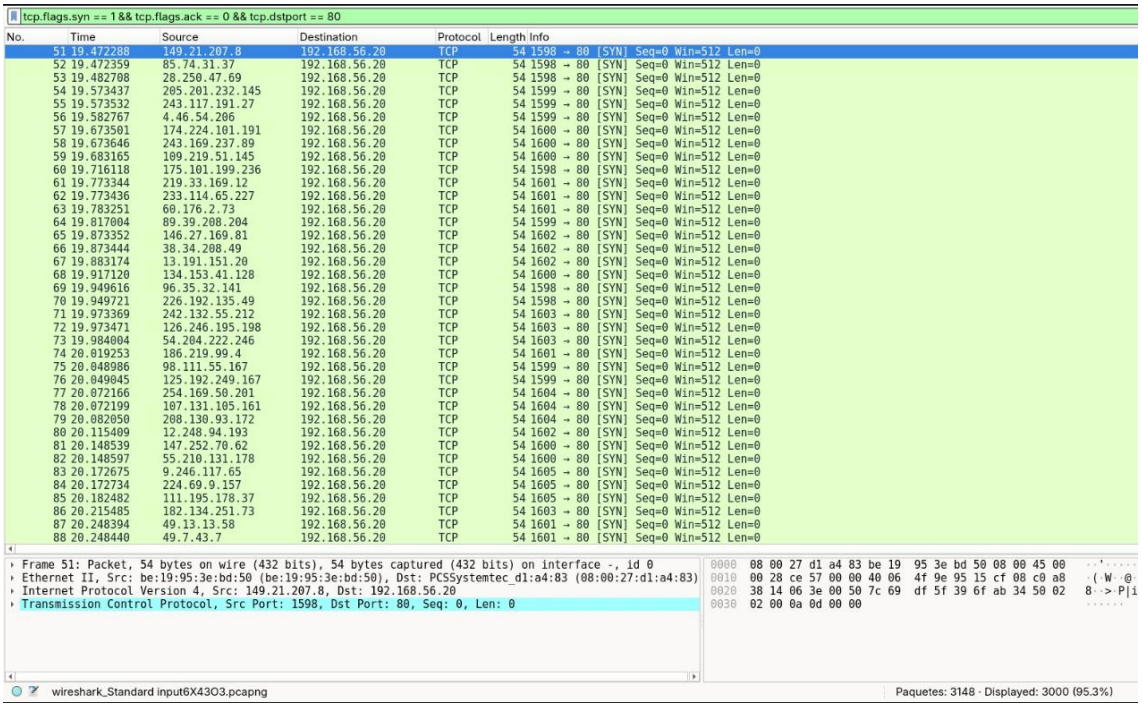
Como se pudo observar el ataque se dirigió al puerto TCP 80, en donde cada bot generó 200 paquetes SYN, logrando así establecer un volumen esperado de paquetes SYN de la siguiente forma.

$$15\ atacantes \times 200\ solicitudes\ = 3000\ paquetes\ SYN\ maliciosos$$

Dentro de esta prueba se realizó una observación destinada a utilizar un filtro con la finalidad de poder identificar paquetes con bandera SYN activa, bandera ACK desactivada y finalmente el destino al puerto 80 que se estableció con anterioridad. Durante la realización se pudo llegar a observar que la captura mostró 3000 paquetes SYN enviados hacia el servidor web como se lo detalla en la siguiente Figura 120.

Figura 120

Generación de solicitudes TCP con atacantes



Además de llegar a denotar métricas importantes como retransmisiones TCP: 0, paquetes RST: 0, CPU: 5,6 % y Memoria: 1,4 %. Con dicha observación se puede llegar a mencionar que el ataque no produjo retransmisiones ni paquetes de reinicio visibles en la captura. Sin embargo, el consumo de CPU del controlador aumentó notablemente pasando de 0,7 % en la prueba normal a 5,6 % durante el ataque. Este aumento mostró que el controlador tuvo que procesar una mayor cantidad de eventos relacionados con nuevos intentos de conexión. Al no existir un sistema de seguridad, las solicitudes SYN no fueron identificadas como maliciosas ni bloqueadas.

Prueba TCP mixta

a) Objetivo de la prueba

La prueba TCP mixta tuvo como finalidad analizar el comportamiento del servidor web cuando coexistían solicitudes HTTP legítimas y un ataque SYN Flood.

b) Configuración y ejecución

En este caso de igual forma se llegó a establecer un levantamiento simultáneo entre 15 usuarios normales y 15 atacantes.

Figura 121

Configuración y ejecución TCP mixta



Los usuarios ejecutaron solicitudes HTTP mediante CURL hacia 192.168.56.20, mientras que los atacantes enviaron paquetes SYN hacia el puerto 80 del mismo servidor. En Wireshark se contabilizaron 3000 solicitudes HTTP GET legítimas con la debida respuesta de 3000 respectivamente. Esto se lo puede evidenciar claramente mediante las siguientes Figuras 122 y 123

Figura 122

Generación de solicitudes TCP mixta

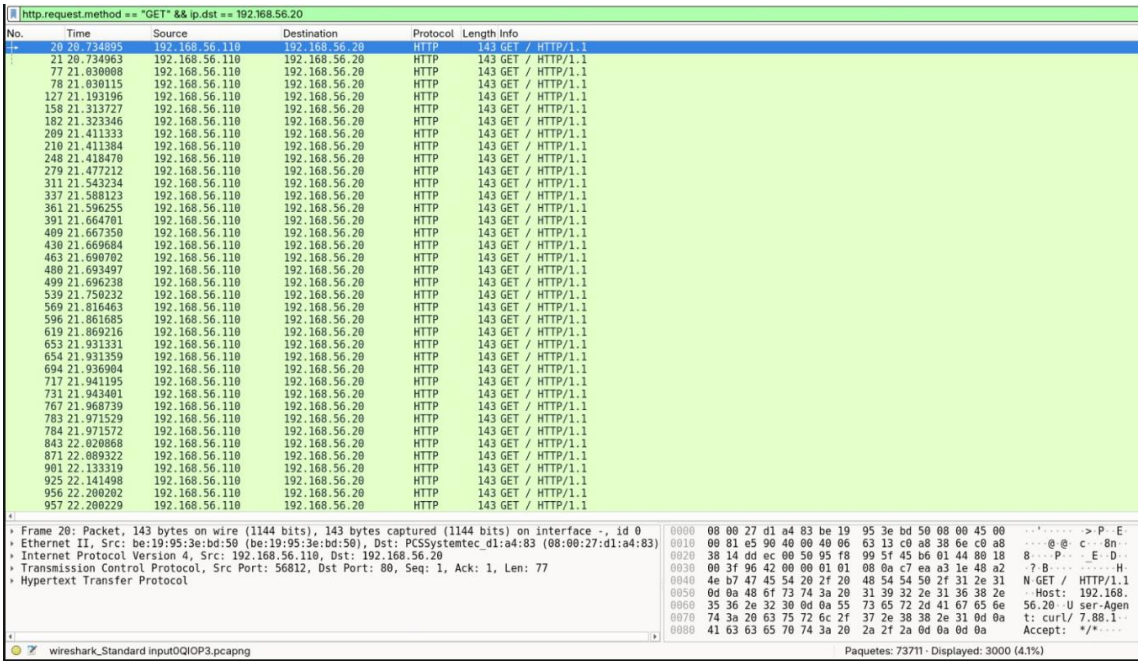
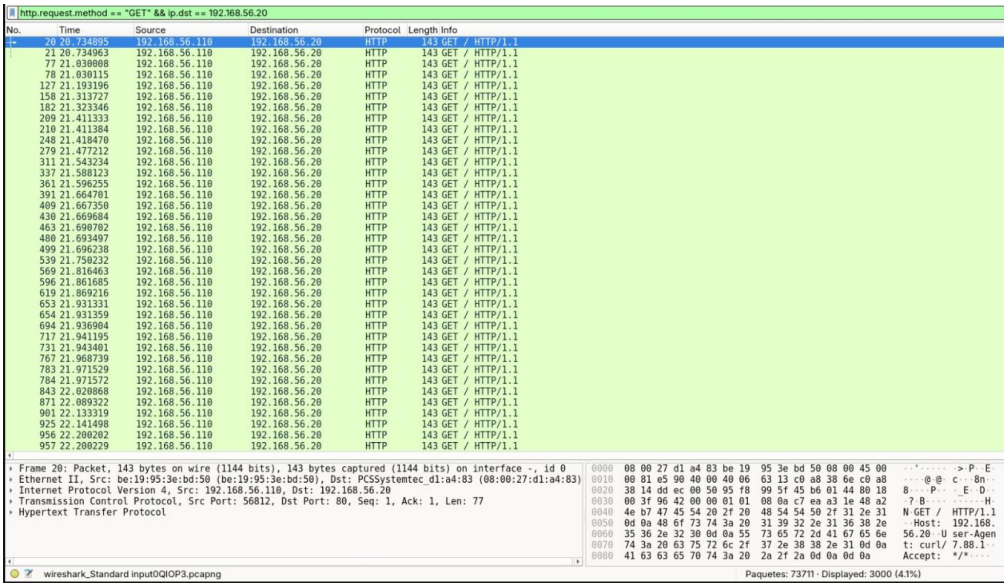


Figura 123

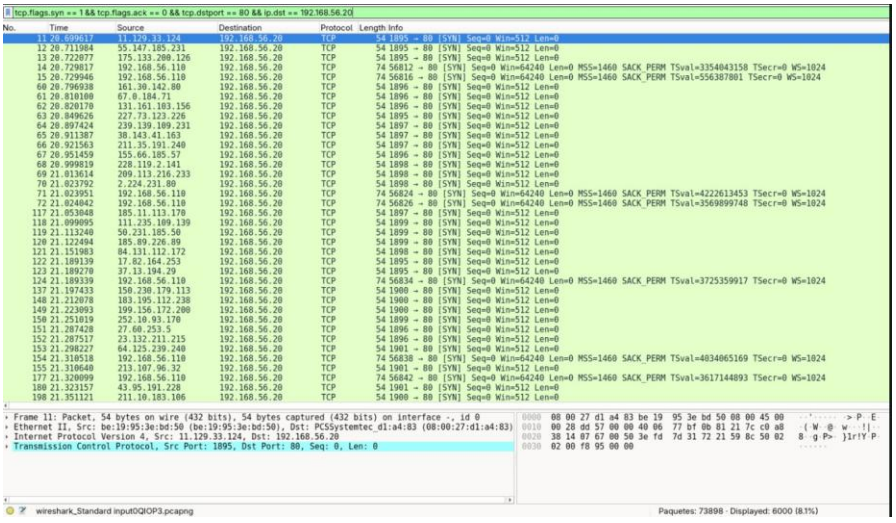
Captura de respuestas de las solicitudes TCP mixta



Logrando así denotar que se registraron 6000 paquetes SYN totales dirigidos hacia el servidor, en donde básicamente se constituyen de paquetes SYN utilizados por los usuarios para iniciar conexiones TCP legítimas y paquetes SYN generados por el ataque.

Figura 124

Captura de paquetes SYN utilizados por los usuarios para iniciar conexiones TCP legítimas y paquetes SYN generados por el ataque



La coexistencia de tráfico normal y malicioso incrementó el uso de CPU del controlador de 0,7 % a 6,8 %. Además, se registró una retransmisión y una disponibilidad del 80 %, lo que evidenció una degradación del servicio y una mayor carga de la infraestructura, aunque las conexiones no fueron cerradas abruptamente.

Prueba UDP con llamada normal

a) Objetivo de la prueba

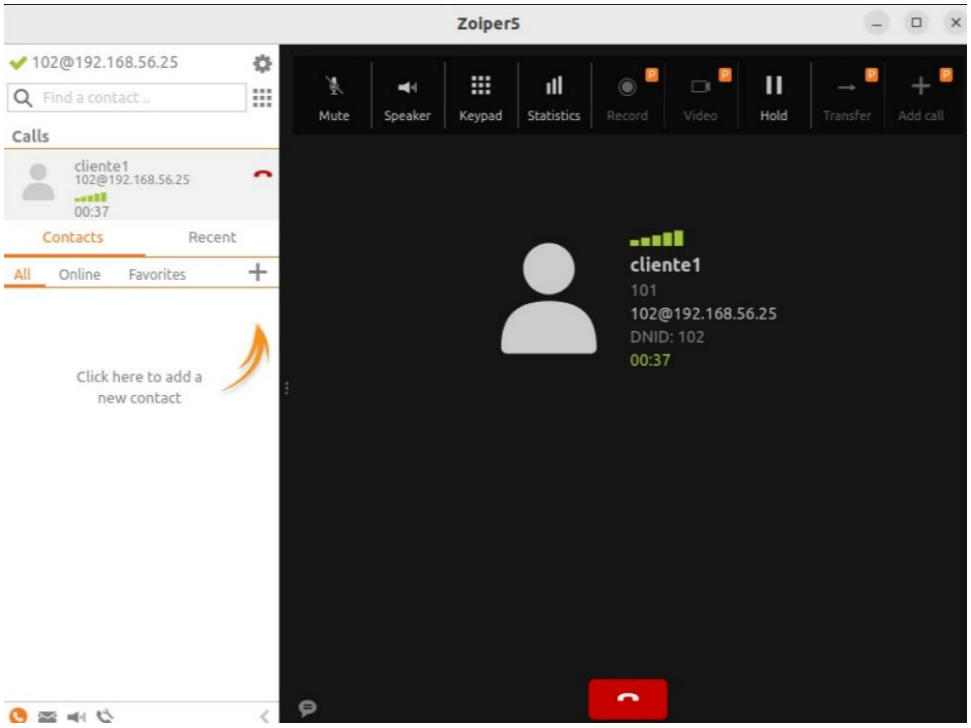
El caso PA-ES1-UDP-04 tuvo como finalidad establecer los valores normales de una llamada VoIP sin presencia de tráfico malicioso. Se analizaron principalmente pérdida de paquetes RTP, delta, jitter y consumo de recursos.

b) Configuración y ejecución

Se utilizaron dos clientes Zoiper registrados en el servidor SIP con dirección 192.168.56.25 como se lo puede observar en la Figura 125. Después de verificar el registro, se estableció una llamada entre las extensiones configuradas.

Figura 125

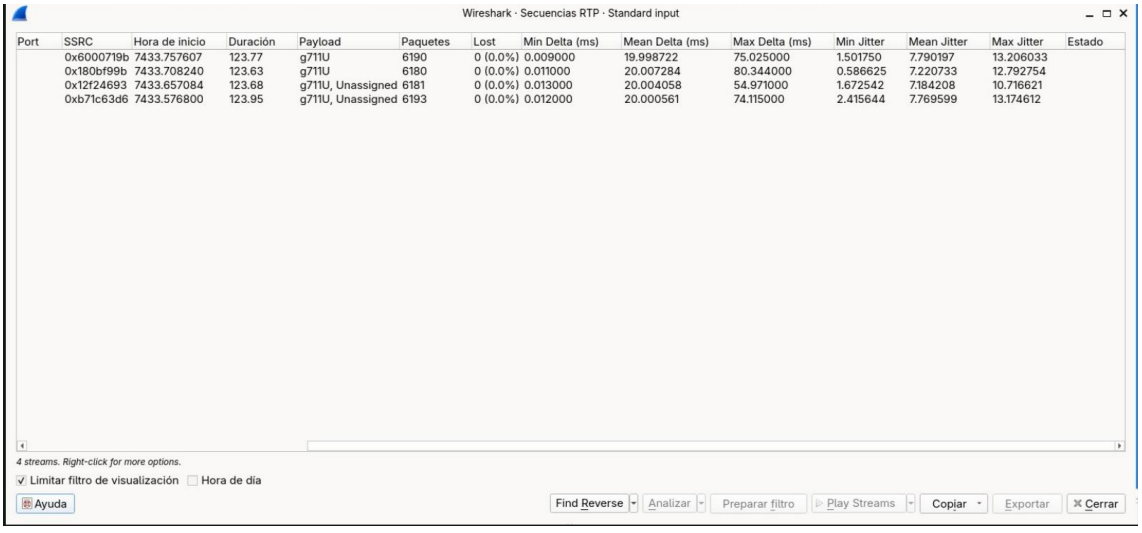
Configuración y ejecución UDP con llamada normal



Durante la llamada se inició una captura en Wireshark y posteriormente se accedió a la herramienta de análisis de flujos RTP.

Figura 126

Captura Wireshark UDP con llamada normal



Wireshark identificó cuatro flujos RTP con aproximadamente 6180 a 6193 paquetes cada uno, sin pérdidas ni errores de secuencia. El delta promedio se mantuvo cercano a 20 ms, mientras que el jitter promedio osciló entre 7,18 y 7,79 ms. Además, el controlador registró un consumo de 0,3 % de CPU y 1,4 % de memoria. Estos resultados evidenciaron una transmisión de audio continua y estable, por lo que se establecieron como línea base del servicio VoIP.

Prueba UDP con llamada bajo ataque

a) Objetivo de la prueba

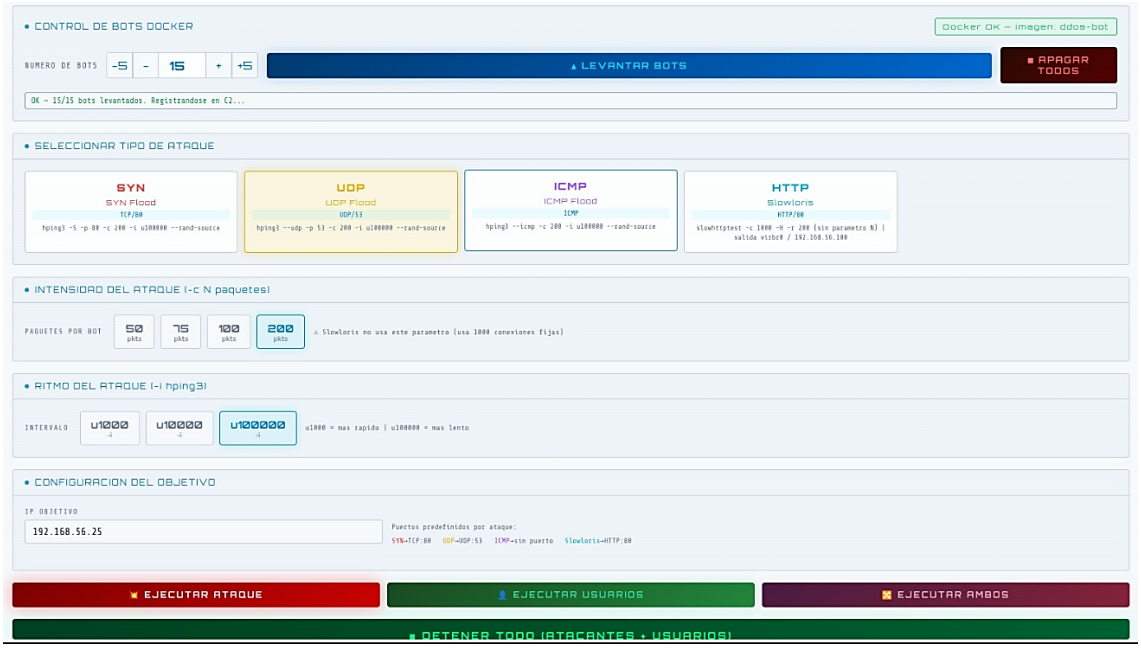
El caso PA-ES1-UDP-05 tuvo como propósito determinar el efecto de un UDP Flood sobre una llamada VoIP activa. La prueba buscó observar si el ataque producía pérdida de paquetes, errores de secuencia, incremento del jitter y mayor consumo de CPU.

b) Configuración y ejecución

Se mantuvo activa la llamada entre los dos clientes Zoiper y de forma simultánea desde el dashboard se seleccionó el ataque UDP Flood.

Figura 127

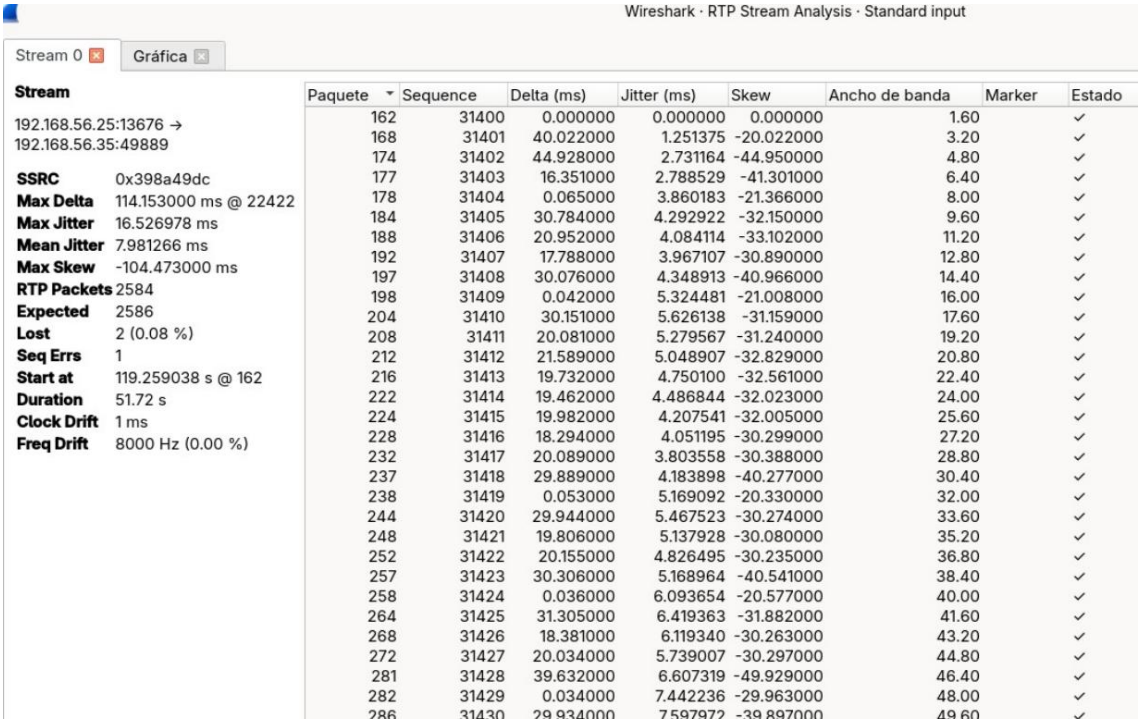
Configuración y ejecución UDP con llamada bajo ataque



Como se pudo observar en la Figura 127 el objetivo configurado fue el 192.168.56.25, la cual llega a corresponder a la dirección al servidor de voz IP. De acuerdo con esto se establecieron 200 paquetes por bot y se ejecutó el tráfico malicioso mientras Wireshark capturaba la llamada.

Figura 128

Captura Wireshark de la llamada bajo ataque



Paquete	Sequence	Delta (ms)	Jitter (ms)	Skew	Ancho de banda	Marker	Estado
162	31400	0.000000	0.000000	0.000000	1.60		✓
168	31401	40.022000	1.251375	-20.022000	3.20		✓
174	31402	44.928000	2.731164	-44.950000	4.80		✓
177	31403	16.351000	2.788529	-41.301000	6.40		✓
178	31404	0.065000	3.860183	-21.366000	8.00		✓
184	31405	30.784000	4.292922	-32.150000	9.60		✓
188	31406	20.952000	4.084114	-33.102000	11.20		✓
192	31407	17.788000	3.967107	-30.890000	12.80		✓
197	31408	30.076000	4.348913	-40.966000	14.40		✓
198	31409	0.042000	5.324481	-21.008000	16.00		✓
204	31410	30.151000	5.626138	-31.159000	17.60		✓
208	31411	20.081000	5.279567	-31.240000	19.20		✓
212	31412	21.589000	5.048907	-32.829000	20.80		✓
216	31413	19.732000	4.750100	-32.561000	22.40		✓
222	31414	19.462000	4.486844	-32.023000	24.00		✓
224	31415	19.982000	4.207541	-32.005000	25.60		✓
228	31416	18.294000	4.051195	-30.299000	27.20		✓
232	31417	20.089000	3.803558	-30.388000	28.80		✓
237	31418	29.889000	4.183898	-40.277000	30.40		✓
238	31419	0.053000	5.169092	-20.330000	32.00		✓
244	31420	29.944000	5.467523	-30.274000	33.60		✓
248	31421	19.806000	5.137928	-30.080000	35.20		✓
252	31422	20.155000	4.826495	-30.235000	36.80		✓
257	31423	30.306000	5.168964	-40.541000	38.40		✓
258	31424	0.036000	6.093654	-20.577000	40.00		✓
264	31425	31.305000	6.419363	-31.882000	41.60		✓
268	31426	18.381000	6.119340	-30.263000	43.20		✓
272	31427	20.034000	5.739007	-30.297000	44.80		✓
281	31428	39.632000	6.607319	-49.929000	46.40		✓
282	31429	0.034000	7.442236	-29.963000	48.00		✓
286	31430	29.934000	7.597972	-39.897000	49.60		✓

Dentro de los resultados observados por la captura se pudo identificar el análisis RTP con las siguientes valoraciones respectivamente.

- Paquetes recibidos: 2584.
- Paquetes esperados: 2586.
- Paquetes perdidos: 2.
- Pérdida: 0,08 %.
- Errores de secuencia: 1.
- Delta máximo: 114,153 ms.
- Jitter máximo: 16,527 ms.
- Jitter promedio: 7,981 ms.
- Duración analizada: 51,72 segundos.

El ataque UDP supuso la pérdida de dos paquetes RTP y la producción de errores de secuencia. Esto pone de manifiesto que la calidad del flujo de voz se vio afectada. El delta máximo alcanzó 114,153 ms, y también se detuvo el jitter a valores importantes, lo que implicó un mayor retraso y desorden en la llegada de los paquetes. Solamente el consumo de CPU del controlador pasó del 0,3 % al 8,6 %; por el contrario, la memoria se estabilizaba en un 1,4 %, lo que demuestra que ese tipo de ataque afecta notablemente la capacidad de procesamiento.

4.1.4.2 Resultados

Las pruebas confirmaron que la red SDN se desempeñó adecuadamente cuando las condiciones del tráfico eran normales; sin embargo, sí presentó degradación ante ataques. Evidentemente, el tráfico malicioso aumentó el consumo del CPU del controlador, disminuyó la disponibilidad del servicio TCP e impactó la comunicación VoIP mediante pérdida de paquetes, errores de secuencia, delta y jitter. Aunque no se pudo observar una interrupción total del servicio, la ausencia de algoritmos de detección y mitigación, a excepción del controlador de ataques, permitió, a su vez, que los ataques impactasen sobre el rendimiento de la red.

4.1.4.3 Segundo escenario: red con sistema de seguridad

En un segundo escenario, el sistema ya instalado en el controlador Ryu fue capaz de detectarlos y bloquearlos correctamente para ataques ICMP y SYN Flood Shore no aislados y permitir tráfico ICMP legítimo con un número de falsos positivos aceptable. Sin embargo, tenían limitaciones en la clasificación del tráfico TCP y UDP ya que identificaron erróneamente comunicaciones legítimas y no tuvieron una buena detección de eventos maliciosos durante la ejecución del ataque UDP. Estas deficiencias generaron una elevada tasa de falsos positivos y coincidieron con la degradación del servicio VoIP, cuya pérdida de paquetes aumentó del 2,4 % al 12 %.

Tabla 27

Resultados obtenidos de la prueba de segundo escenario: red con sistema de seguridad

Protocolo	Condición evaluada	Resultados obtenidos	MTTD obtenido	Interpretación
ICMP	15 usuarios normales	Se analizaron 178 eventos. Los 178 fueron clasificados como normales, obteniendo una exactitud del 100 %. La confianza promedio fue del 88,16 %.	0,0000 s	No se generaron bloqueos ni alertas. El sistema permitió correctamente todo el tráfico legítimo.
ICMP	15 atacantes	Se analizaron 530 eventos ICMP Flood. Los 530 fueron identificados correctamente, con una exactitud del 100 % y confianza promedio del 99,97 %.	0,9118 s	Los 530 eventos fueron bloqueados. El sistema detectó y mitigó correctamente el ataque ICMP.
ICMP	15 usuarios normales y 15 atacantes	Se registraron 574 eventos: 518 internos y 56 externos. Los 56 eventos externos fueron identificados como ataque y 512 de los 518 internos como normales. Se produjeron 6 falsos positivos y ningún falso negativo. La exactitud general fue del 98,95 %.	0,7140 s	Se bloquearon los 56 eventos externos. Los 6 eventos internos clasificados como ataque fueron enviados a análisis manual. El sistema detectó todos los ataques y mantuvo una baja tasa de falsos positivos del 1,16 %.

Protocolo	Condición evaluada	Resultados obtenidos	MTTD obtenido	Interpretación
TCP	15 usuarios normales	Se analizaron 2112 eventos legítimos. El modelo clasificó 2059 como TCP normal, 52 como SYN Flood y 1 como UDP Flood. Por tanto, TCP normal representa el 97,49 % de las muestras evaluadas.	0,1553 s	Se registró una tasa de acierto del 97.49 % como TCP normal y Solo 53 eventos legítimos fueron clasificados erróneamente como ataque, lo que representa una tasa de falsos positivos del 2,51 %.
TCP	15 atacantes	Se analizaron 174 eventos SYN Flood. Los 174 fueron identificados correctamente, alcanzando una exactitud del 100 % y una confianza promedio del 99,90 %.	0,6003 s	Todos los eventos fueron bloqueados. El sistema respondió correctamente ante el ataque TCP ejecutado de forma aislada.
TCP	15 usuarios normales y 15 atacantes	Se registraron 2446 eventos durante la prueba TCP mixta, de los cuales 234 correspondieron a tráfico externo asociado al ataque y 2212 a tráfico interno generado por usuarios legítimos. El sistema	0,4180 s	La sensibilidad para detectar ataques fue del 100 %, debido a que los 234 eventos externos asociados al SYN Flood fueron

Protocolo	Condición evaluada	Resultados obtenidos	MTTD obtenido	Interpretación
		identificó correctamente los 234 eventos externos como SYN Flood. En cuanto al tráfico interno, 2200 eventos fueron clasificados correctamente como TCP normal, mientras que 12 eventos legítimos fueron clasificados erróneamente como SYN Flood.		identificados correctamente. Asimismo, la especificidad frente al tráfico normal fue del 99,46 %, ya que 2200 de los 2212 eventos internos fueron clasificados como TCP normal y Solo 12 eventos legítimos fueron identificados erróneamente como SYN Flood, lo que representa una tasa de falsos positivos del 0,54 %
UDP	Llamada normal	Se analizaron 1708 eventos. Un total de 1694 fueron clasificados como normales, 12 como UDP Flood y 2 como ICMP Flood. Únicamente el 0,70 % fue identificado específicamente	2,2 s	La llamada presentó un nivel de calidad de aceptable, jitter indicado de 0 ms y una pérdida de paquetes del 2,4 %. El resultado

Protocolo	Condición evaluada	Resultados obtenidos	MTTD obtenido	Interpretación
		como UDP Flood y el 99,18 % permaneció clasificado como normal.		evidencia que parte del tráfico RTP legítimo fue confundido con ataques con un 0.70% de falso positivos. Aunque se aplicaron acciones de bloqueo sobre los eventos clasificados como ataque, la llamada presentó degradación.
UDP	Llamada con ataque	Se analizaron 4297 eventos. El modelo clasificó 3550 como normales, 675 como UDP Flood y 72 como ICMP Flood. La exactitud para reconocer tráfico normal fue del 82,62% y para tráfico anómalo fue de 17,38%.	3,1 s	La afectación de la calidad aumentó aproximadamente de 2,4 % en condición normal a 12 % durante el ataque, debido a que el tráfico UDP/RTP requiere baja latencia y transmisión continua. Por ello, el procesamiento del controlador y del

Protocolo	Condición evaluada	Resultados obtenidos	MTTD obtenido	Interpretación
				modelo puede no ser lo suficientemente rápido para evitar pérdidas o variaciones en tiempo real.

Nota: en las pruebas mixtas de ICMP y TCP se utilizó el campo ip_scope para diferenciar los eventos en las direcciones externas correspondieron al tráfico generado por los atacantes y las direcciones internas al tráfico de los usuarios legítimos. Esto permitió calcular los verdaderos positivos, falsos positivos y la exactitud general de manera más coherente.

4.1.4.4 Tercer escenario: sistema de seguridad externo

En el tercer escenario se evaluó el sistema de seguridad ubicado externamente al controlador, por lo que su función se concentró en observar y detectar el tráfico sin intervenir directamente en el reenvío de los paquetes. Los resultados muestran que los tiempos de detección fueron generalmente superiores a los obtenidos cuando el sistema se encontraba integrado en el controlador.

Tabla 28

Resultados obtenidos de la prueba de tercer escenario: red con sistema de seguridad externo

Protocolo	Condición evaluada	MTTD obtenido	Interpretación
ICMP	15 usuarios normales	0,0000 s	No se registró detección de ataque, lo cual fue consistente con la presencia exclusiva de tráfico legítimo.

Protocolo	Condición evaluada	MTTD obtenido	Interpretación
ICMP	15 atacantes	2,7204 s	El sistema externo identificó el ICMP Flood, pero necesitó aproximadamente 2,72 segundos para generar la detección.
ICMP	15 usuarios normales y 15 atacantes	0,7790 s	La detección fue más rápida que en la prueba realizada únicamente con atacantes, lo que indica que la concentración y variación del tráfico mixto facilitó la identificación del comportamiento anómalo.
TCP	15 usuarios normales	0,6406 s	Se registró un valor de detección durante una prueba normal. Este resultado puede estar relacionado con una clasificación falsa positiva del tráfico HTTP legítimo, comportamiento que también fue observado en el segundo escenario.
TCP	15 atacantes	2,2498 s	El ataque TCP fue reconocido después de aproximadamente 2,25 segundos. El tiempo fue superior al obtenido con el sistema integrado en el controlador.
TCP	15 usuarios normales y 15 atacantes	0,4720 s	Fue el menor MTTD registrado para TCP. El sistema externo identificó rápidamente la presencia de tráfico

Protocolo	Condición evaluada	MTTD obtenido	Interpretación
			anómalo cuando coexistieron usuarios y atacantes.
UDP	Llamada normal	3,5000 s	El registro de un MTTD durante la llamada normal debe interpretarse con precaución, debido a que podría estar relacionado con falsos positivos sobre el tráfico RTP legítimo.
UDP	Llamada con ataque	3,7082 s	Fue el mayor tiempo de detección registrado en el tercer escenario. El sistema necesitó más de 3,7 segundos para identificar el ataque UDP dirigido al servicio VoIP.

En comparación con el sistema integrado, la ubicación externa presentó un incremento del tiempo de detección en los ataques ejecutados de manera aislada. Para ICMP, el MTTD aumentó de 0,9118 a 2,7204 segundos, mientras que para TCP pasó de 0,6003 a 2,2498 segundos. Esto evidencia que la integración directa con el controlador favoreció una identificación más rápida de los ataques, debido a que el análisis se realizó dentro del propio plano de control y no requirió enviar la información hacia un componente externo.

Una vez ejecutados el segundo y el tercer escenario, se compararon los tiempos medios de detección (MTTD) obtenidos en cada prueba, con el propósito de determinar cuál configuración respondió con mayor rapidez ante los ataques. Esta comparación resultó necesaria debido a que la ubicación del sistema de seguridad influyó directamente en el proceso de captura, procesamiento y análisis de los eventos. Mientras que en el segundo escenario el sistema estuvo integrado en el

controlador Ryu, en el tercero funcionó de manera externa, lo que permitió identificar si la transferencia adicional de información incrementó el tiempo requerido para detectar el tráfico malicioso, como se lo puede observar en la siguiente Tabla 29.

Tabla 29

Comparación de MTTD entre el escenario 2 y 3

Protocolo	Condición evaluada	MTTD escenario 2	MTTD escenario 3	Comparación
ICMP	15 usuarios normales	0,0000 s	0,0000 s	No existe diferencia entre los escenarios, debido a que no se detectaron ataques durante el tráfico legítimo.
ICMP	15 atacantes	0,9118 s	2,7204 s	El escenario 3 necesitó 1,8086 s más que el escenario 2 para detectar el ICMP Flood. Por tanto, el sistema integrado presentó una respuesta más rápida.
ICMP	15 usuarios normales y 15 atacantes	0,7140 s	0,7790 s	El escenario 3 registró un tiempo 0,0650 s mayor que el escenario 2. Ambos presentaron tiempos similares, aunque el sistema integrado fue ligeramente más rápido.
TCP	15 usuarios normales	0,1553 s	0,6406 s	El escenario 3 presentó un tiempo 0,4853 s mayor. Sin embargo, estos valores se relacionan con posibles falsos positivos sobre el tráfico TCP/HTTP legítimo y no con una detección real de ataque.
TCP	15 atacantes	0,6003 s	2,2498 s	El escenario 3 necesitó 1,6495 s más para detectar el SYN Flood. El sistema integrado respondió con mayor rapidez al procesar los eventos directamente en el controlador.

Protocolo	Condición evaluada	MTTD escenario 2	MTTD escenario 3	Comparación
TCP	15 usuarios normales y 15 atacantes	0,4180 s	0,4720 s	El escenario 3 registró un MTTD 0,0540 s mayor que el escenario 2. La diferencia fue reducida, aunque el sistema integrado mantuvo una respuesta ligeramente más rápida.
UDP	Llamada normal	2,2000 s	3,5000 s	El escenario 3 presentó un tiempo 1,3000 s mayor. Estos valores deben interpretarse con precaución, pues corresponden a posibles falsos positivos sobre tráfico RTP legítimo.
UDP	Llamada con ataque	3,1000 s	3,7082 s	El escenario 3 necesitó 0,6082 s más que el escenario 2 para identificar el ataque UDP. Aunque el sistema integrado fue más rápido, ambos presentaron limitaciones frente al tráfico UDP/RTP.

Nota. Los valores de MTTD corresponden al promedio de tres ejecuciones realizadas para cada condición evaluada. La diferencia se calculó restando el MTTD del escenario 2 al MTTD del escenario 3

4.2 Análisis de los resultados con el método STRIDE

La metodología STRIDE permitió clasificar las amenazas identificadas en la arquitectura SDN de acuerdo con seis categorías, suplantación de identidad, manipulación de información, repudio, divulgación de información, denegación de servicio y elevación de privilegios. Su aplicación se centró en los principales activos del entorno experimental, el controlador Ryu, los switches Open vSwitch, el servidor web, el servidor VoIP, los usuarios legítimos, los registros del sistema y el módulo de detección.

Los resultados mostraron que la amenaza predominante fue la denegación de servicio, debido a que los ataques ICMP Flood, SYN Flood y UDP Flood incrementaron la carga del controlador y afectaron el funcionamiento de los servicios. También se identificaron indicios relacionados con suplantación de identidad, debido al empleo de múltiples direcciones IP externas durante los ataques. Las demás categorías no fueron reproducidas directamente en el plan de pruebas, sin embargo, se analizaron considerando los componentes y evidencias disponibles.

Tabla 30

Análisis de los resultados de las pruebas de los diferentes escenarios con el método STRIDE

Categoría STRIDE	Activos relacionados	Evidencia obtenida	Análisis del impacto	Nivel observado
Suplantación de identidad	Controlador Ryu, switches Open vSwitch, servidor web, servidor VoIP y usuarios legítimos	Durante los ataques ICMP Flood y SYN Flood se observaron paquetes provenientes de múltiples direcciones IP externas.	La variación de las direcciones de origen puede dificultar la identificación del atacante y el bloqueo mediante reglas estáticas. El sistema integrado logró reconocer y bloquear los flujos externos en las pruebas ICMP y SYN Flood; sin embargo, no se evaluaron mecanismos específicos de autenticación de dispositivos o validación de direcciones IP.	Medio
Manipulación de información	Reglas OpenFlow, configuraciones del controlador, modelo XGBoost, paquetes y registros	No se observaron modificaciones no autorizadas de paquetes, reglas OpenFlow, configuraciones, archivos del modelo o registros del sistema.	Los ataques ejecutados estuvieron orientados a la saturación de recursos y no a modificar información. Por ello, no existe evidencia experimental suficiente para confirmar una amenaza de manipulación dentro de las pruebas realizadas.	Bajo / no comprobado

Categoría STRIDE	Activos relacionados	Evidencia obtenida	Análisis del impacto	Nivel observado
Repudio	Registros del controlador, dashboard, archivos CSV y registros de bloqueo	Los eventos fueron almacenados con fecha, hora, dirección IP, protocolo, predicción, confianza y acción aplicada.	Los registros permiten relacionar cada evento con su clasificación y con la respuesta ejecutada, aportando trazabilidad. No obstante, no se evaluaron firmas digitales, protección criptográfica ni mecanismos de integridad de los logs.	Bajo a medio
Divulgación de información	Dashboard, archivos CSV, Redis, topología y métricas del sistema	Los registros contienen direcciones IP, puertos, protocolos, valores de entropía, predicciones y acciones de mitigación. No se ejecutaron ataques orientados a extraer esta información.	No se evidenció divulgación de información durante las pruebas. Sin embargo, el acceso no autorizado a los registros o al dashboard podría revelar datos técnicos útiles para reconocer la estructura y el funcionamiento de la red.	Bajo / potencial
Denegación de servicio	Controlador Ryu, servidor web, servidor VoIP, switches y usuarios legítimos	En el escenario sin seguridad se observaron incrementos de CPU, intermitencia ICMP, degradación TCP y afectación de la llamada VoIP. El sistema integrado detectó correctamente los ataques ICMP Flood y SYN Flood aislados, pero presentó falsos positivos en	Fue la amenaza de mayor impacto. Los ataques incrementaron la carga del controlador y afectaron la disponibilidad y calidad de los servicios. Los falsos positivos también pueden generar una denegación de servicio indirecta al restringir tráfico legítimo, mientras que los	Alto

Categoría STRIDE	Activos relacionados	Evidencia obtenida	Análisis del impacto	Nivel observado
		TCP y una detección insuficiente del UDP Flood. El sistema externo registró mayores tiempos de detección.	falsos negativos permiten que el ataque continúe afectando la red.	
Elevación de privilegios	Sistema operativo del controlador, aplicaciones Ryu, dashboard, Docker y archivos del modelo	No se observaron intentos de obtener permisos administrativos, ejecutar comandos privilegiados o modificar componentes internos del sistema.	Las pruebas se limitaron a la generación de tráfico normal y malicioso. No se evaluaron vulnerabilidades de acceso, escalamiento de privilegios o compromiso administrativo del controlador.	Bajo / no comprobado

4.2.1.1 Análisis de la denegación de servicio por escenario

La categoría Denial of Service fue el eje central del análisis, ya que los ataques buscaron degradar la disponibilidad de la red y sus servicios. En el primer escenario, la ausencia de protección permitió que el tráfico malicioso afectara directamente al controlador y a los servicios ICMP, HTTP y VoIP, incrementando el consumo de CPU, reduciendo la disponibilidad y provocando pérdidas, errores de secuencia y variaciones de delta y jitter. En cuanto al segundo contexto, se comprobó que el sistema integrado detectó y bloqueó con precisión los ataques ICMP y TCP, por lo que la sensibilidad fue del 100 % y la exactitud alcanzó un valor cercano a 99,51 % en la evaluación TCP mixta. Sin embargo, UDP obtuvo un elevado porcentaje de falsos positivos que alcanzó el 17,38 % que provocó degradación de la llamada VoIP. En el tercer contexto, el sistema externo logró identificar los ataques con MTTD los tiempos más elevados: 2,7204 segundos para ICMP, 2,2498 segundos para TCP y 3,7082 segundos para UDP, resultando que la localización del sistema externo contribuyó a elevar el tiempo de recolección y análisis de los eventos.

4.2.1.2 Valoración general mediante STRIDE

La utilización de STRIDE hizo posible determinar que la principal vulnerabilidad de la arquitectura de la misma era la disponibilidad. El sistema integrado fue capaz de mitigar el impacto de ataques ICMP y SYN Flood, dado que su detección y bloqueo de flujos maliciosos era más rápida en comparación con el sistema externo. Sin embargo, los falsos positivos observados en TCP y la baja detección del UDP Flood demuestran que el mecanismo debe ser ajustado antes de ser considerado como fiable para una eventual implementación operativa.

La suplantación de direcciones de origen también debe considerarse, debido a que el uso de múltiples IP externas dificulta la identificación de los atacantes mediante reglas estáticas. En

cambio, no se obtuvieron evidencias directas de manipulación, divulgación de información o elevación de privilegios, porque dichas amenazas no formaron parte de las pruebas realizadas. En términos generales, el análisis STRIDE muestra que el sistema aporta una mejora frente a la denegación de servicio cuando se encuentra integrado al controlador, principalmente para ICMP y TCP SYN Flood aislados. Sin embargo, la protección debe ir acompañada de ajustes en el modelo, validación VoIP de tráfico, disminución de los falsos positivos y controles de autenticación, integridad de registros y acceso a los componentes administrativos.

Conclusiones

El repaso del estado del arte ha permitido establecer que los ataques DDoS de baja intensidad suponen una amenaza suficiente para las redes definidas por software a partir de la capacidad que poseen para degradar progresivamente los servicios sin provocar saltos bruscos del tráfico. También se estableció que la entropía de Rényi aporta información sobre la concentración y dispersión de los flujos, mientras que XGBoost y SHAP permiten combinar capacidad de clasificación e interpretabilidad. De esta manera, se cumplió el primer objetivo específico relacionado con el análisis de técnicas de detección basadas en entropía y aprendizaje automático interpretable.

El escenario sin sistema de seguridad permitió establecer una línea base del comportamiento de la infraestructura. Normalizando condiciones se tuvo 0 % de pérdida en el tráfico ICMP, donde se enviaron 3000 solicitudes y se recibieron 3000 respuestas. En el TCP normal, el servidor web obtuvo una correcta respuesta a 3000 solicitudes HTTP GET, obteniendo así una disponibilidad del 100 %, con 6 retransmisiones TCP y 0 paquetes RST. En el UDP normal, la llamada VoIP no presentó perdida de paquetes RTP y el controlador presentó un consumo aproximado de 0,3 % de CPU, así como 1,4 % de memoria. Sin embargo, al incorporar tráfico malicioso se registraron incrementos en el consumo de CPU, intermitencia en las comunicaciones y degradación del servicio de voz, evidenciando que la ausencia de mecanismos de protección permite que los ataques afecten directamente al controlador y a los servicios de la red.

La integración del sistema de seguridad en el controlador presentó resultados favorables frente a ataques ICMP. En la prueba normal se reconocieron con exactitud los 178 eventos normativos, y a su vez en el caso del ICMP Flood fueron reconocidos y bloqueados con exactitud los 530 eventos maliciosos. En la prueba mixta se reconocieron los 56 eventos externos asociados

al ataque, mientras que entre los 518 eventos internos se generaron unos escasos seis falsos positivos, logrando así una exactitud general aproximada de 98,95 %. Estos resultados demuestran que las características extraídas, incluida la entropía, permitieron distinguir bien el tráfico normal y malicioso relativo al ICMP.

En el ataque SYN Flood entregado de forma individual el sistema pudo detectar y bloquear de forma certera hasta un total de 174 Eventos analizados, lo que da lugar a detecciones totalmente satisfactorias en un 100 %. En la evaluación del tráfico TCP normal se concretaron 2112 eventos legítimos, de los cuales 2059 fueron cifrados como TCP normal, un 97,49 %; 52 eventos fueron clasificados como SYN Flood, lo que equivale a un 2,46 %; y 1 evento fue clasificado como UDP Flood, correspondiente a un 0,05 %. En la prueba TCP mixta los eventos fueron 2446 eventos, es decir, 234 externos al ataque y 2212 internos desde los usuarios legítimos. El sistema detectó de forma certera los 234 eventos externos como SYN Flood y, además, 2200 eventos internos como TCP normal, para sólo tener un total de 12 falsos positivos. Esto representa una sensibilidad del 100 %, una especificidad del 99,46 %, una tasa de falsos positivos del 0,54 % y una exactitud binaria global aproximada del 99,51 %.

Los resultados del protocolo UDP mostraron la principal debilidad del sistema. En la llamada normal se analizaron 4297 eventos, de los cuales 3550 fueron clasificados como normales, equivalentes al 82,62 %. Sin embargo, 675 eventos fueron clasificados como UDP Flood, correspondientes al 15,71 %, y 72 eventos como ICMP Flood, equivalentes al 1,68 %. En total, fueron erróneamente clasificados como tráfico de ataque un total de 747 eventos legítimos, lo que representa un 17,38 % de tasa de falsos positivos. En la llamada UDP Flood se procesaron 1708 eventos, de los cuales se identificaron 1694 como normales, 12 como UDP Flood y 2 como ICMP Flood. A pesar de que se llevaron a cabo bloqueos en los eventos clasificados como ataque, la

llamada VoIP tuvo un deterioro, siendo del 12 % aproximadamente durante el ataque. Esto demuestra que el modelo requiere una mayor representación de tráfico RTP y UDP en el conjunto de entrenamiento, especialmente por la velocidad de transmisión y la baja latencia requerida en comunicaciones en vivo.

La ubicación del sistema de seguridad dentro del controlador ofreció tiempos de detección menores que su implementación externa. Para ICMP Flood, el MTTD aumentó de 0,9118 segundos en el sistema integrado a 2,7204 segundos en el sistema externo. El tiempo para SYN Flood aumentó de 0,6003 a 2,2498 segundos. En UDP el sistema externo registró un MTTD de 3,7082 segundos. La diferencia nuevamente pone de manifiesto la fuerte influencia de la integración directa con el plano de control para un MTTD bajo. En ese sentido, un MTTD bajo no garantiza por sí solo el óptimo funcionamiento del sistema, ya que este debe ser evaluado juntamente con la ratio de falsos positivos y negativos.

La evaluación realizada mediante STRIDE categorizó que la categoría de más elevado impacto fue la de Denial of Service, pues los ataques que se procederán a evaluar aquí afectaron la disponibilidad, el procesamiento del controlador y la calidad de los servicios web junto con los de VoIP. Este hecho se evidenció en el incremento del uso del procesador (CPU), la intermitencia en ICMP mixto, la reducción de la disponibilidad en TCP mixto y la degradación de la llamada VoIP tras realizar la prueba de ataque UDP. Asimismo, el análisis de riesgos también identificó riesgos potenciales de suplantación de direcciones IP, acceso a los registros y protección ante configuraciones. Las categorías de manipulación, divulgación de información y elevación de privilegios no fueron comprobadas experimentalmente, debido a que el plan de pruebas se concentró en ataques de denegación de servicio.

Recomendaciones

Ampliar el conjunto de datos utilizado para el entrenamiento incorporando una mayor cantidad y diversidad de tráfico TCP legítimo, solicitudes HTTP normales, comunicaciones SIP y flujos RTP. Esto permitirá que el modelo diferencie con mayor precisión entre un servicio utilizado intensivamente y un ataque, reduciendo los falsos positivos observados en TCP y mejorando la detección de UDP Flood.

Comprobar y revisar el proceso de etiquetado de datos previo a hacer un nuevo entrenamiento. Deberá evitarse tener archivos duplicados con distintos valores de ground_truth, deberá validarse que cada una de las grabaciones se corresponda con el escenario indicado y deberá separarse correctamente el tráfico legítimo del tráfico que generan los atacantes. La calidad de las etiquetas es muy importante para evitar que el modelo aprenda asociaciones incorrectas.

Evaluar modelos de clasificación jerárquica o específicos por protocolo. En una primera etapa se podría identificar si el tráfico corresponde a ICMP, TCP o UDP y, posteriormente, aplicar un clasificador entrenado específicamente para cada protocolo. Esta separación reduciría la confusión entre tráfico HTTP normal, HTTP Flood, SYN Flood y comunicaciones RTP.

Ajustar los parámetros de las ventanas de observación, el orden de la entropía de Rényi, los umbrales de confianza y el número de ventanas consecutivas requeridas para confirmar un ataque. Tomar decisiones basadas en múltiples observaciones en cadena puede ayudar a que las variaciones temporales del tráfico legítimo no conllevan errores en el caso de los bloqueos.

Incorporar gradualmente las políticas de mitigación. Los eventos con alta confianza o de origen externo pueden ser bloqueados directamente, mientras que los eventos de controversia o de origen interno pueden ser sujetos a limitación de la tasa de manera ascendente o ser monitorizados adicionalmente, o ser sometidos a un análisis manual. También se sugiere la utilización de listas

de confianza para servidores y servicios internos con el objetivo de evitar el bloqueo del tráfico legítimo.

Utilizar SHAP de manera sistemática para estudiar los falsos positivos y falsos negativos obtenidos durante las pruebas. El análisis debe identificar qué características provocaron que el tráfico HTTP normal fuera clasificado como ataque y por qué la mayoría de los eventos UDP Flood fueron reconocidos como normales. También se debe comprobar cuantitativamente la contribución de la entropía de Rényi en cada tipo de clasificación.

Se recomienda mantener constantes la cantidad de usuarios y atacantes, el número de paquetes, la duración, la tasa de transmisión y los recursos asignados, para obtener resultados comparables y reducir la influencia de variaciones ocasionales.

Cruzaremos el modelo de sesión, utilizando un tiempo base de muestreo del tráfico para separar la información de entrenamiento, validación y prueba bien por sesiones, fechas, nodos o bien escenarios completos, en lugar de realizar simplemente una aleatorización de los registros. Esto permitirá evaluar de manera más fidedigna la capacidad real de generalización del modelo a tráfico no observado y reducir el riesgo a que flujos de tráfico similares queden simultáneamente en entrenamiento y test.

Mejorar la arquitectura del sistema externo aprovechando el procesamiento local, reduciendo el intervalo de consulta y la transmisión asíncrona de eventos. Estas mejoras pueden llegar a reducir el MTTD observado respecto al sistema integrado. Para entornos donde la velocidad de respuesta sea un objetivo sustancial, se recomienda mantener la inferencia suficientemente cerca del controlador y utilizar el sistema externo de auditoría, almacenamiento o análisis complementario.

Antes de desplegar la propuesta en una red productiva, realizar una validación piloto con tráfico real y comparar XGBoost con otros modelos, como Random Forest, con árboles de decisión y esquemas de ensamble. La selección final tiene que ir acompañada de otros parámetros que además de evaluar exactitud, precisión, recall y F1-score, también consideren MTTD, la tasa de falsos positivos, el consumo de recursos, la explicabilidad y la continuidad de los servicios de tráfico legítimo.

Se recomienda implementar una arquitectura de balanceo de carga entre controladores SDN, donde un controlador puede actuar como maestro y otro esclavo. Este modelo permitiría realizar un mejor balanceo del procesamiento de los eventos, equilibrar mejor el esfuerzo en la detección o en la carga individual de un controlador en el procesamiento. Además, en caso de que el controlador principal presente una falla o interrupción, el controlador secundario podría asumir sus funciones, garantizando mayor disponibilidad, tolerancia a fallos y continuidad operativa del sistema de detección.

Bibliografía

Agrafiotis, I., Nurse, J. R. C., Goldsmith, M., Creese, S., & Upton, D. (2018). A taxonomy of cyber-harms: Defining the impacts of cyber-attacks and understanding how they propagate. *Journal of Cybersecurity*, *4*(1), tyy006. https://doi.org/10.1093/cybsec/tyy006

Agrawal, N., & Tapaswi, S. (2021). An SDN-Assisted Defense Mechanism for the Shrew DDoS Attack in a Cloud Computing Environment. *Journal of Network and Systems Management*, *29*(2), 12. https://doi.org/10.1007/s10922-020-09580-7

Alashhab, A. A., Zahid, M. S. M., Abdullahi, M., & Rahman, M. S. (2023). Real-Time Detection of Low-Rate DDoS Attacks in SDN-Based Networks Using Online Machine Learning Model. *2023 7th Cyber Security in Networking Conference, CSNet 2023*, 95–101. https://doi.org/10.1109/CSNet59123.2023.10339791

Benzekki, K., El Fergougui, A., & El Belrhiti El Alaoui, A. (2017). Software-defined networking (SDN): A survey. *Security and Communication Networks*, *9*. https://doi.org/10.1002/sec.1737

Chai, K. Y., & Zolkipli, M. F. (2021). *View of Review on Confidentiality, Integrity and Availability in Information Security*. Journal of ICT in Education. https://ejournal.upsi.edu.my/index.php/JICTIE/article/view/5203/3091

Fioravanti, G., Spina, M. G., & Rango, F. De. (2023). Entropy based DDoS Detection in Software Defined Networks. *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, 636–639. https://doi.org/10.1109/CCNC51644.2023.10059949

Gcore. (2024). *Key takeaways DDoS Attack Trends for Q1-Q2 2024 DDoS peak attack power continues to surge The number of attacks is growing*.

Haleplidis, E., Pentikousis, K., Denazis, S., Salim, J. H., Meyer, D., & Koufopavlou, O. (2015). *Software-Defined Networking (SDN): Layers and Architecture Terminology* (Número 7426). RFC Editor. https://doi.org/10.17487/RFC7426

Índice 1. OpenFlow y sus herramientas 2. Software Defined Networking (SDN) 3. Network Function Virtualization (NFV). (s/f).

Institute of Standards, N. (2024). *The NIST Cybersecurity Framework (CSF) 2.0*. https://doi.org/10.6028/NIST.CSWP.29

Joosten, R., & Nieuwenhuis, L. J. (2017). *Comparing Alternatives to Measure the Impact of DDoS Attack Announcements on Target Stock Prices*. www.abhishta.xyz/

Kreutz, D., Ramos, F. M. V, Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, *103*(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Lara, A., Kolasani, A., & Ramamurthy, B. (2014a). Network Innovation using OpenFlow: A Survey. *IEEE Communications Surveys & Tutorials*, *16*(1), 493–512. https://doi.org/10.1109/SURV.2013.081313.00105

Lara, A., Kolasani, A., & Ramamurthy, B. (2014b). Network Innovation using OpenFlow: A Survey. *IEEE Communications Surveys & Tutorials*, *16*(1), 493–512. https://doi.org/10.1109/SURV.2013.081313.00105

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, *30*.

Mohammad Aladaileh Mohammed Anbar, I. H. H. Y. K. S. Y.-W. C. (2021). Entropy-Based Approach to Detect DDoS Attacks on Software Defined Networking Controller. *Computers, Materials & Continua*, *69*(1), 373–391. https://doi.org/10.32604/cmc.2021.017972

Nadeem, A. ;, Vos, D. ;, Cao, C. ;, Pajola, L. ;, Dieck, S. ;, Baumgartner, R. ;, & Verwer, S. (2023). *SoK: Explainable Machine Learning for Computer Security Applications; SoK: Explainable Machine Learning for Computer Security Applications*. 221–240. https://doi.org/10.1109/EuroSP57164.2023.00022

Open Networking Foundation. (2013). *Software-Defined Networking: The New Norm for Networks - Open Networking Foundation*. https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/

Puranik, K., Patil, K., Ghaligi, G., Jannu, R., Patil, S., Narayan, D. G., & Kachavimath, A. (2023). A Two-level DDoS attack Detection using Entropy and Machine Learning in SDN. *2023 3rd International Conference on Intelligent Technologies (CONIT)*, 1–7. https://doi.org/10.1109/CONIT59222.2023.10205776

Reddy, K. B., & Meera, S. (2024). DDoS attack detection in cloud using ensemble model tuned with optimal hyperparameter. *International Journal of Adaptive Control and Signal Processing*, *38*(5), 1594–1620. https://doi.org/https://doi.org/10.1002/acs.3766

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*(5), 206–215. https://doi.org/10.1038/s42256-019-0048-x

Scott-Hayward, S., Natarajan, S., & Sezer, S. (2016). A Survey of Security in Software Defined Networks. *IEEE Communications Surveys & Tutorials*, *18*(1), 623–654. https://doi.org/10.1109/COMST.2015.2453114

Scott-Hayward, S., O'Callaghan, G., & Sezer, S. (2013). Sdn Security: A Survey. *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 1–7. https://doi.org/10.1109/SDN4FNS.2013.6702553

SDN and Openflow For Beginners With Hands On Labs PDF | PDF | Network Switch | Port (Computer Networking). (s/f). Recuperado el 2 de julio de 2025, de https://es.scribd.com/document/416270120/SDN-and-Openflow-for-beginners-with-hands-on-labs-pdf?utm_source=chatgpt.com

Sudar, K. M., & Deepalakshmi, P. (2022). Flow-Based Detection and Mitigation of Low-Rate DDOS Attack in SDN Environment Using Machine Learning Techniques. En P. Nayak, S. Pal, & S.-L. Peng (Eds.), *IoT and Analytics for Sensor Networks* (pp. 193–205). Springer Singapore.

Tríada de la CIA: definición y ejemplos — Wallarm. (s/f). Recuperado el 11 de junio de 2025, de https://lab.wallarm.com/what/definicion-de-la-triada-de-la-cia-ejemplos-de-confidencialidad-integridad-y-disponibilidad/?lang=es

Unidas, N. (2030). *La Agenda 2030 y los Objetivos de Desarrollo Sostenible: una oportunidad para América Latina y el Caribe*. www.issuu.com/publicacionescepal/stacks

Uysal, I., & Kose, U. (2024). Analysis of Network Intrusion Detection via Explainable Artificial Intelligence: Applications with SHAP and LIME. *2024 Cyber Awareness and Research Symposium (CARS)*, 1–6. https://doi.org/10.1109/CARS61786.2024.10778742

Von Solms, R., & Van Niekerk, J. (2013). From information security to cyber security. *Computers & Security*, *38*, 97–102. https://doi.org/10.1016/J.COSE.2013.04.004

Yu, H., Yang, W., Cui, B., Sui, R., & Wu, X. (2024). Renyi entropy-driven network traffic anomaly detection with dynamic threshold. *Cybersecurity*, *7*(1), 64. https://doi.org/10.1186/s42400-024-00249-1

Zhang, N., Jaafar, F., & Malik, Y. (2019). Low-Rate DoS Attack Detection Using PSD Based Entropy and Machine Learning. *Proceedings - 6th IEEE International Conference on Cyber*

Security and Cloud Computing, CSCloud 2019 and 5th IEEE International Conference on Edge Computing and Scalable Cloud, EdgeCom 2019, 59–62. https://doi.org/10.1109/CSCloud/EdgeCom.2019.00020

Anexos

Anexo 1: Trabajos relacionados

Ficha Bibliográfica 1	
Autores	Puranik et al.
Editorial	IEEE
Título	A Two-level DDoS attack Detection using Entropy and Machine Learning in SDN
Año	2023
Tipo de entropía	Shannon

El objetivo del trabajo “A Two-level DDoS attack Detection using Entropy and Machine Learning in SDN” fue proponer un enfoque de dos niveles para la detección de ataques DDoS en entornos SDN, donde en un primer nivel se aplica un cálculo de entropía de Shannon sobre flujos de tráfico para identificar comportamientos sospechosos de manera temprana y, en un segundo nivel, se emplean algoritmos de aprendizaje automático (Naive Bayes, SVM y Random Forest) para confirmar la detección y reducir la tasa de falsos positivos, obteniendo como resultado que el algoritmo Random Forest alcanzó una exactitud del 98.3%.

Ficha Bibliográfica 2	
Autores	Fioravanti et al.
Editorial	IEEE

Título	Entropy based DDoS Detection in Software Defined Networks
Año	2023
Tipo de entropía	Shannon

El objetivo del trabajo “Entropy-based DDoS Detection in Software Defined Networks” fue diseñar un mecanismo de detección de ataques DDoS en entornos SDN utilizando la entropía de Shannon como métrica central, aplicada sobre las direcciones IP de destino contenidas en los mensajes *PACKET IN*. Los autores plantean que, durante un ataque, la concentración de tráfico hacia una sola víctima provoca una caída significativa en el valor de la entropía. Para mejorar la precisión de la detección, el estudio introduce parámetros de configuración como el tamaño de la ventana de observación (W), el umbral de entropía y el número de ventanas consecutivas (N) en las que se mantiene la anomalía, proponiendo un proceso de ajuste fino (tuning) que permita reducir tanto falsos positivos como falsos negativos.

Ficha Bibliográfica 3

Autores	Mohammad Aladaileh Mohammed Anbar
Editorial	IEEE
Título	Entropy-Based Approach to Detect DDoS Attacks on Software Defined Networking Controller
Año	2021
Tipo de entropía	Rényi

El estudio “Entropy-Based Approach to Detect DDoS Attacks on Software Defined Networking Controller” propuso un esquema de detección basado en la entropía conjunta de Rényi con umbral dinámico, utilizando dos características de cabecera de los paquetes (IP de origen y de destino) para identificar tanto ataques DDoS de baja como de alta tasa en escenarios con una o múltiples fuentes y objetivos, implementados en Mininet con el controlador POX. Los resultados mostraron altas tasas de detección y bajos falsos positivos en comparación con enfoques basados en entropía de Shannon.

Ficha Bibliográfica 4

Autores	Zhang et al.
Editorial	IEEE
Título	Low-Rate DoS Attack Detection Using PSD based Entropy and Machine Learning
Año	2019
Tipo de entropía	PSD (Power Spectral Density)

El estudio “Low-Rate DoS Attack Detection Using PSD Based Entropy and Machine Learning” propone un detector para LDoS que primero calcula entropía basada en la densidad espectral de potencia (PSD-entropy) y la compara con umbrales adaptativos para filtrar rápido muestras obvias, y luego usa SVM con selección/reducción de características (PCA) para los casos ambiguos, alcanzando 99.19% de detección con buena eficiencia.

Anexo 2: Repositorio del sistema desarrollado

El código fuente correspondiente al sistema de detección de ataques DDoS de baja intensidad en redes SDN, junto con los scripts utilizados para la generación de tráfico, procesamiento de datos, entrenamiento del modelo, implementación del controlador y visualización de resultados, se encuentra disponible en el siguiente repositorio de GitHub:

Repositorio: https://github.com/0Dynamoxd/TesisDDoS