Detección y Mitigación de Ataques de Denegación de Servicio en Redes IoT Usando Inteligencia Artificial (IA) y Técnicas de Aprendizaje Automático (ML)

Trabajo Fin de Grado

Ingeniería de Tecnologías y Servicios de Telecomunicación.

Universitat Oberta de Catalunya

Josep López Capó

Área de Administración de Redes y Sistemas Operativos

Ámbito de Seguridad en Redes de Computadores

Tutor

Fernando Pérez López

Junio 2024

Resumen

El Internet de las Cosas (IoT) se presenta como la red de dispositivos inteligentes que surgen en respuesta a la necesidad de integrar nuevos servicios (transporte, atención médica, infraestructura, etc) en prácticamente todos los ámbitos de nuestras vidas. En consecuencia, esta rápida proliferación de internet ha ofrecido nuevas oportunidades a los usuarios atacantes. La forma más común, por su sencillez, de afectar una red, es mediante el ataque de denegación de servicio. Dicho ataque, consiste, en esencia, en copar de tráfico malicioso una red entorpeciendo o anulando el servicio ofrecido. Por ende, la implementación de mecanismos efectivos de defensa se establece como una necesidad básica en toda red de Internet. En el presente trabajo se utiliza las técnicas de aprendizaje automático (ML) usando inteligencia artificial (IA) con el fin de implementar un Sistema de Detección de Intrusos (IDS) para detectar y mitigar supuestos Ataques de Denegación de Servicio (DoS). Este trabajo también tiene como objetivo evaluar dos modelos de aprendizaje automático para la detección de ofensivas de denegación de servicio DIS-Flood en un entorno IoT simulado. Concretamente, se estudian los algoritmos *Support Vector Machine* (SVM) y *Random Forest* (RF) sobre el conjunto de datos IoTR – DS. La tasa de exactitud obtenida para SVM es de 98.6%, mientras que la de *Random Forest* es de 99.4%. La tasa de precisión fue de 98.2% en SVM y 99.2% en RF. La sensibilidad fue de 98.7% en SVM en comparación con 99.6% de RF. Los resultados muestran que *Random Forest* superó a *Support Vector Machine* en todas las métricas anteriores presentándose como la opción más adecuada para su implementación en el IDS.

Palabras Clave — Internet, IoT, Ataques de Denegación de Servicio, Inteligencia Artificial, Aprendizaje profundo, Aprendizaje Automático, Ataque, Ciberdelincuente, Modelo de Aprendizaje, Clasificador, Support Vector Machine, Random Forest, Matriz de Confusión, IDS.

Abstract

The Internet of Things (IoT) emerges as the network of smart devices that arise in response to the need to integrate new services (transportation, healthcare, infrastructure, etc.) into virtually every aspect of our lives. Consequently, this rapid proliferation of the internet has offered new opportunities to attacking users. The most common, due to its simplicity, way to affect a network is through a denial of service attack. This attack consists, essentially, in flooding a network with malicious traffic, hindering or nullifying the offered service. Therefore, the implementation of effective defense mechanisms is established as a basic necessity in every internet network. In this work, machine learning (ML) techniques using artificial intelligence (AI) are employed to implement an Intrusion Detection System (IDS) to detect and mitigate alleged Denial of Service Attacks, as well as to discuss the evolution of the results. This thesis aims to evaluate two machine learning models for the detection of DIS-Flood Denial of Service Attacks in a simulated IoT environment. Support Vector Machine (SVM) and Random Forest (RF) classifiers are used on the IoTR – DS dataset [Resources 9]. The accuracy rate obtained for SVM is 98.6%, while for Random Forest it is 99.4%. The precision rate was 98.2% in SVM and 99.2% in RF. Sensitivity was 98.7% in SVM compared to 99.6% in RF. The results show that Random Forest outperformed Support Vector Machine in all the aforementioned metrics, presenting itself as the most suitable option for implementation in attack detection.

Keywords — Internet, IoT, Denial of Service Attacks, Artificial Intelligence, Deep Learning, Machine Learning, Attack, Cybercriminal, Learning Model, Classifier, Support Vector Machine, Random Forest, Confusion Matrix, IDS.

Índice de Contenidos

Índice de Figuras 5
Índice de Tablas 6
Cuerpo de la Memoria 8
Capítulo I. Introducción 8
1. Justificación, punto de partida y aportación 8
2. Motivación 9
3. Objetivo 10
4. Enfoque y metodología seguida 10
5. Diagrama de Gantt 11
6. Producto obtenido 12
7. Breve descripción de la estructura de la memoria 12
Capítulo II. Fundamentos de IoT y Ataques de Denegación de Servicio 12
1. Privacidad y seguridad 15
2. Fundamentos de Ataques de Denegación de Servicio 16
3. Trayectoria DoS y DDoS 17
4. Tipos de ataques en Capa de Transporte 19
4.1 Ataques de tipo inundación TCP 19
4.2 Ataques de tipo inundación UDP 19
4.3 Ataques de tipo inundación ICMP 20
5. Tipos de ataques en Capa de Aplicación 20
5.1 Aprovechamiento método GET de solicitudes HTTP 20
5.2 Solicitudes incompletas de HTTP con POST 21
5.3 Ataque de dirección duplicada 22
5.4 Ofensiva a través del enrutador 22
6. Análisis de tendencias de ataque en capa de aplicación 22
Capítulo III. Fundamentos de Inteligencia Artificial 23
1. Aprendizaje Automático (ML) y Aprendizaje Profundo (DL) 26
2. Diferencias entre ML y DL 26
3. Extracción de características 34
4. Construcción del Modelo 34
5. Desafíos para sistemas inteligentes basados en ML y DL 36
Capítulo IV. Simulación e implementación IDS 37
1. Metodología y herramientas 37
2. Simulación DoS - UDP flood 38
2.1 Cliente 39
2.2 Servidor 40

2.3 Atacante 41
2.4 Escenario 42
2.5 Análisis resultados UDP flood 43
3. Simulación DoS – DIS flood 44
3.1 Nodo Sink 46
3.2 Cliente 48
3.3 Malicioso 50
4. Análisis resultados DIS flood 55
5. Implementación IDS 56
6. Módulo de Colección 58
6.1 Características de la Capa Física 58
6.2 Características de la Capa de Red 58
6.3 Características de la Capa de Aplicación 58
6.4 Selección de Características 58
7. Módulo de Clasificación y Detección 59
7.1 Fase de Pre-aprendizaje 60
7.1.1 Support Vector Machine (SVM) 60
7.1.2 Arboles de Decisión (Random Forest) 60
7.1.3 Verificación de Algoritmo 61
7.2 Fase de Post-aprendizaje 62
8. IDS o Agente IDS 63
9. Exportación de Datos en Tiempo Real 64
9.1 Módulo de Manejo de Datos 64
9.2 Módulo de Visualización de Datos 65
9.3 Módulo de Gestión de Base de Datos 66
9.4 Módulo API 66
10. Colección de Datos a Tiempo Real 67
11. Configuración del modelo de aprendizaje 69
11.1 Máquina de Vectores de Soporte (SVM) 69
11.2 Bosques Aleatorios 73
12. Entrenamiento y validación. 73
12.1 Entrenamiento 74
12.2 Validación 75
13. Código de entreno 75
Capítulo V. Análisis de resultados 77
1. Evaluación SVM 77
2. Evaluación Bosque Aleatorio (Random Forest) 78
3. Estudio final 79

Capítulo VI. Contribuciones y Valoración económica 80
1. Marco de protección usando ML 80
2. Metodología Inteligente 80
3. Valoración económica 80
Capítulo VII. Conclusión 81
1. Limitaciones y líneas futuras de trabajo 82
2. Reflexión final 83
Glosario de Abreviaturas 84
Bibliografía 85
Webgrafía 87
Recursos 87
Anexo 89

Índice de Figuras

Figura 1: Evolución del uso de dispositivos conectados a IoT. Años 2019 - 2024, con previsiones de 2024 a 2030 (en billones) Fuente: [Webgrafía 6]. 13
Figura 2: Arquitectura IoT 14
Figura 3: Estructura general de trabajo de IoT 14
Figura 4: Escenarios de IoT 15
Figura 5: Ataque SYN flood 18
Figura 6: Ataque DDoS de Capa de Aplicación, distribución por país. Fuente: [Webgrafía 7]. 18
Figura 7: Ataque de inundación ICMP 20
Figura 8: Comparativa ciclos HTTP ante ataque 21
Figura 9: Funcionamiento mecanismo DAD. 22
Figura 10: Frecuencia de uso de protocolos de capa de aplicación en la realización de ataques DDoS durante el período de Q1-2019 a Q1-2021 Fuente: [Webgrafía 20]. 23
Figura 11: Principio de razonamiento basado en casos 25
Figura 12: Conceptos y clases de algoritmos de aprendizaje automático 27
Figura 13: Flujo de operaciones basado en Fog Computing 31
Figura 14: Distancia euclidiana entre dos puntos Fuente: [Webgrafía 8] 32
Figura 15: Clasificación básica según el protocolo k-Nearest Neighbors 32
Figura 16: Reducción dimensional en SOM. Fuente: [Webgrafía 9]. 33
Figura 17: Código cliente.c 39
Figura 18: Código servidor.c 40
Figura 19: Código atacante.c 41
Figura 20: Creación motas 42
Figura 21: Escenario ataque DoS 42

Figura 22: Inicio proceso nodo atacante 43
Figura 23: Inicio proceso nodo servidor 43
Figura 24: Inicio proceso nodo atacante 43
Figura 25: Operaciones de proceso 43
Figura 26: Mensajes enviados 44
Figura 27: Creación Nodo Sink simulación DIS flood 46
Figura 28: Creación Nodo Cliente simulación DIS flood 48
Figura 29: Creación Nodo Malicioso simulación DIS flood 50
Figura 30: Topología de red simulación DIS flood 51
Figura 31: Distribución de nodos....51
Figura 32: Nodo Sink (1), nodo malicioso (12) y nodos normales (2-11) 52
Figura 33: Creación de las conexiones con el nodo Sink 52
Figura 34: Tiempos de proceso y conexión 53
Figura 35: Configuración colección de datos 54
Figura 36: Envío de datos 54
Figura 37: Consumo medio de energía de los nodos antes del ataque 55
Figura 38: Consumo medio de energía de los nodos después del ataque 55
Figura 39: Esquema general IDS 56
Figura 40: Fase pre-aprendizaje 57
Figura 41: Fase post-aprendizaje....57
Figura 42: Proceso selección de características....59
Figura 43: Módulo de Clasificación y Detección 60
Figura 44: Código general manejo datos entrantes al sensor 65
Figura 45: Marco de Exportación de Datos API REST 66
Figura 46: Salida JSON 67
Figura 47: Esquema físico de ubicación de red e IDS....67
Figura 48: Módulo Colección de Datos....68
Figura 49: Esquema algorítmico de colección de datos....68
Figura 50: Diagrama SVM 70
Figura 51: Código de entreno 76
Figura 52: Matriz de confusión SVM 77
Figura 53: Matriz de confusión Random Forest....78

Índice de Tablas

Tabla 1: Diferencias generales entre DoS y DDoS....17
Tabla 2: Comparativa métodos GET y POST. 21

Tabla 3: Tipos de aprendizaje automático. 30
Tabla 4: Principales arquitecturas de aprendizaje profundo (DL). 36
Tabla 5: Características de la simulación. 45
Tabla 6: Características de nodo. 46
Tabla 7: Características extraídas. 59
Tabla 8: Subsets obtenidos 74
Tabla 9: Comparaciones exactitud. 78
Tabla 10: Comparaciones Precisión. 78
Tabla 11: Comparaciones Recall. 79
Tabla 12: Comparaciones F-Measure. 79
Tabla 13: Coste Humano. 81
Tabla 14: Coste Material. 81

Todas las fuentes son de elaboración propia a no ser que se indique lo contrario.

Cuerpo de la Memoria

El presente trabajo se divide en 7 capítulos; tres de ellos, Capitulo I, II, III de carácter introductorio donde se presentan los fundamnetos de los conocimientos que se aplicarán en los Capítulos IV y V. Distinguimos, pues: Capítulo I en donde se introducen las motvaciones de la memoria; Capítulo II, que expone los fundamentos de IoT y Ataques de Denegación de Servicio; Capítulo III, que elabora una introducción sobre los fundamentos de Inteligencia Artificial; Capítulo IV, en donde desarrolla la fase de Simulación e Implementación IDS; Capítulo V, que elabora el análisis de resultados; Capítulo VI que revisa las contribuciones y valoración económica; y, por último, el Capítulo VII que concluye sobre las líneas futuras y elabora una reflexión final del trabajo.

Véase el resumen a continuación:

Capítulo I. Introducción
Capítulo II. Fundamentos de IoT y Ataques de Denegación de Servicio
Capítulo III. Fundamentos de Inteligencia Artificial
Capítulo IV. Simulación e Implementación IDS
Capítulo V. Análisis de resultados
Capítulo VI. Contribuciones y valoración económica
Capítulo VII. Conclusión

Capítulo I. Introducción

1. Justificación, punto de partida y aportación

Los Ataques de Denegación de Servicio (DoS) y Denegación de Servicio Distribuido (DDoS) son ofensivas dedicadas a la privación de servicios dañando los recursos de la red. Se trata de uno de los ataques más comunes y eficaces en Internet presentándose como una amenaza real para el tráfico legítimo entre la relación cliente-servidor. Con el auge del Internet de las cosas (IoT) dichos ataques se han ido amoldando haciéndose más difíciles de detectar y, en consecuencia, más difíciles de rechazar mediante mitigaciones que aseguren la red.

Desde el año 2022 se puede apreciar una fuerte proliferación de la inteligencia artificial que abre nuevos campos de investigación en la ciberseguridad. Aprovechando este contexto de avances como punto de partida, esta memoria respalda evidencias de garantías de seguridad en cuanto a la defensa de redes IoT mediante el uso de inteligencia artificial, aportando nuevas líneas de investigación dedicadas a la prevención de ataques de denegación de servicio. Este estudio trata sobre la detección y mitigación de ataques de denegación de servicio en redes IoT usando inteligencia artificial y técnicas de aprendizaje automático, y pretende aportar 3 distintivos al estado del arte: el desarrollo de

nuevas técnicas de IA, nuevos métodos de detección que cumplan con a) eficacia y b) escalabilidad, y la promoción de un nuevo estándar de seguridad en IoT. Con todo ello, y, en consecuencia, se desarrolla un Sistema de Detección de Intrusos (IDS) que detecta y mitiga ofensivas de denegación de servicio medinate un modelo de *machine learning* entrenado e implementado

2. Motivación

Decía Kevin Mitnick, el pirata informático más famoso de Estados Unidos, que la tecnología avanza rápidamente, y con ella, también las amenazas. Solo estando actualizados podemos sortear las vulnerabilidades del mundo tecnológico en el que estamos sumergidos:

> "Cualquier empresa que se describa a sí misma utilizando la palabra "seguridad" debe parecer un desafío especialmente interesante. Si están involucrados con la seguridad, ¿significa eso que son tan conscientes de la seguridad que no habría forma de penetrar? Para cualquier grupo de personas con mentalidad de hacker, debe parecer un desafío irresistible, especialmente cuando, como en este caso, los individuos no tenían nada más que el nombre de su empresa objetivo para empezar." (Kevin Mitnick, The Art of Intrusion, 2005, 196).

Mitnick nos sugiere que la seguridad se trata de un proceso, no de un producto. La expansión de las tecnologías, abren campo a los delincuentes en los tres aspectos más importantes de la ciberseguridad: la confidencialidad, la disponibilidad y la integridad. Así pues, el presente trabajo se elabora en motivo del cumplimiento de los tres aspectos anteriores aprovechando los últimos avances, que lejos de posicionarse en dirección opuesta, navegan en favor de la ayuda y la seguridad.

Desde el año 2022, vemos una creciente tendencia del desarrollo de herramientas de inteligencia artificial con el propósito de ofrecernos servicios que hasta ahora eran inexistentes —o, por lo menos, su calidad—: elaboración personalizada de contenido multimedia, traducción inmediata de videos, codificación automática, motores de análisis, elaboración de textos, automatización, desarrollo web, etc. No obstante, también encontramos oportunidades para la seguridad que también deben ser explotadas en pos de la salvaguarda de las vulnerabilidades de Internet.

3. Objetivo

Con este trabajo, se pretende analizar los diferentes algoritmos de Inteligencia Artificial (IA) y Aprendizaje Automático (ML) aplicados a la detección y mitigación de ataques de denegación de servicio. Las principales contribuciones se enumeran a continuación: i) presentación y análisis detallado de los ataques de denegación de servicio; ii) descripción IA, aplicación de los métodos ML basados en la detección de ataques DoS/DDoS y desarrollo de un Sistema de Detección de Intrusos; y iii) entreno de modelos de lenguaje y análisis de los resultados para su implementación.

La revisión de dichos medios de protección debe abrazar los tres pilares de la ciberseguridad: **confidencialidad** —datos accesibles por los usuarios oportunos— ,**integridad** —precisión en la información—, y **disponibilidad** —información y servicios disponibles el máximo de tiempo posible.

4. Enfoque y metodología seguida

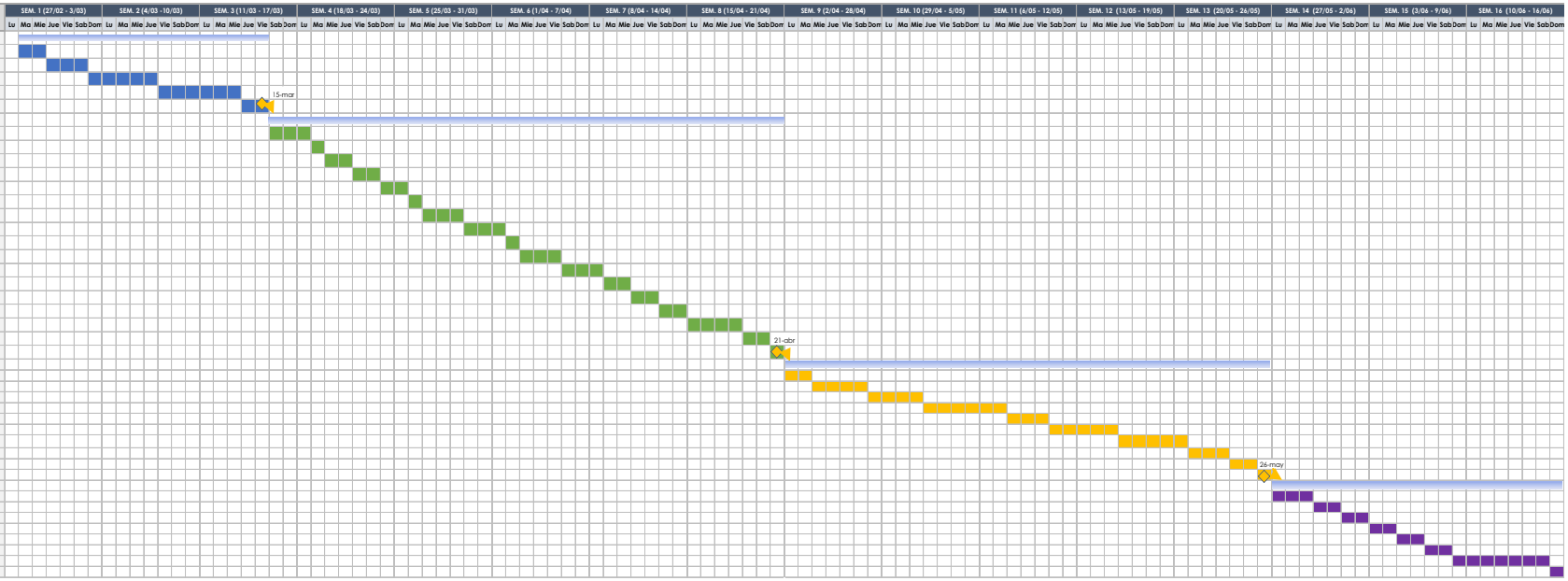
Haremos uso de un enfoque axiomático-deductivo en donde extraeremos proposiciones —aplicaciones—, a partir de un conjunto de premisas. —recopilación de datos—. La metodología seguida es cuantitativa y se centra en el análisis de los mismos. El presente trabajo sigue la siguiente sucesión lineal de eventos: exposición de los fundamentos de IoT, DoS e IA; creación de simulaciones; implementación de Sistema de Detección de Intrusos; y extracción de resultados. Finalmente se reflexiona sobre los mismos.

5. Diagrama de Gantt

PLANIFICACIÓN TRABAJO FINAL DE GRADO

TÍTULO	Detección y Mitigación de Ataques de Denegación de Servicio en Redes IoT Usando Inteligencia Artificial (IA) y Técnicas de Aprendizaje Automático (ML).
ALUMNO	Josep López Capó
PROFESOR	Fernando Pérez López
FECHA	martes, 27 de febrero de 2024

ID	TAREA TÍTULO	INICIO FECHA	FIN FECHA	DÍAS
1	**PEC 1: planificación**	**2024-02-27**	**2024-03-15**	**18**
2	Análisis de información disponible	2024-02-27	2024-02-28	2
3	Consolidar tema	2024-02-29	2024-03-02	3
4	Establecer objetivos y recursos	2024-03-03	2024-03-07	5
5	Redacción Planificación	2024-03-08	2024-03-13	5
6	Revisión y entrega PEC 1	2024-03-12	2024-03-15	3
7	**PEC 2: entrega 40% - 60%**	**2024-03-16**	**2024-04-21**	**37**
9	Redacción Capítulo I	2024-03-16	2024-03-18	3
10	Recopilar información bases IoT	2024-03-19	2024-03-19	1
11	Inicio Redacción Capítulo II	2024-03-20	2024-03-21	2
12	Análisis tendencia de ataque	2024-03-22	2024-03-23	2
13	Clasificación ataques	2024-03-24	2024-03-25	2
14	Fundamentos IA	2024-03-26	2024-03-26	1
15	Inicio Capítulo IA	2024-03-27	2024-03-29	3
16	Sistemas de detección de redes	2024-03-30	2024-04-01	3
17	Corrección errores PEC 1	2024-04-02	2024-04-02	1
18	Estudio de modelos	2024-04-03	2024-04-05	3
19	Inicio fase de simulación	2024-04-06	2024-04-08	3
20	Instalación de CodeLite y VmWare	2024-04-09	2024-04-10	2
21	Configuración Contiki	2024-04-11	2024-04-12	2
22	Preparación entorno	2024-04-13	2024-04-14	2
23	Codificación usuarios	2024-04-15	2024-04-18	4
24	Extracción primeriza de datos	2024-04-19	2024-04-20	2
25	Revisión PEC 2 y entrega	2024-04-20	2024-04-21	1
26	**PEC 3: entrega 60% - 90%**	**2024-04-22**	**2024-05-26**	**35**
27	Corrección PEC 2	2024-04-22	2024-04-23	2
28	Extracción exaustiva de datos	2024-04-24	2024-04-27	4
29	Configuración modelo	2024-04-28	2024-05-01	4
30	Implementación IDS	2024-05-02	2024-05-07	6
31	Fase detección	2024-05-08	2024-05-10	3
32	Fase mitigación	2024-05-11	2024-05-15	5
33	Inicio Capítulo V	2024-05-16	2024-05-20	5
34	Evaluación resultados	2024-05-21	2024-05-23	3
35	Revisión	2024-05-24	2024-05-25	2
36	Eentrega PEC 3	2024-05-26	2024-05-26	1
37	**PEC 4: entrega final**	**2024-05-27**	**2024-06-16**	**21**
38	Inicio Capítulo VI	2024-05-27	2024-05-29	3
39	Conclusión	2024-05-30	2024-05-31	2
40	Reflexión final	2024-06-01	2024-06-02	2
41	Revisión indices y tablas	2024-06-03	2024-06-04	2
42	Revisión de incidencias	2024-06-05	2024-06-06	2
43	Revisión exhaustiva final	2024-06-07	2024-06-08	2
44	Creación presentación y video	2024-06-09	2024-06-15	7
45	Documento de autoevaluación	2024-06-16	2024-06-16	1



6. Producto obtenido

Mediante la memoria obtendremos:

- IDS

  Diseñaremos un Sistema de Detección de Intrusos estructurado en diferentes bloques que monitorearán el flujo de la red IoT con el objetivo de detectar y mitigar ataques; en consecuencia, se elaborará un aviso de acuerdo a la estimación del agente IDS.

- Algoritmo ML

  Estudiaremos la eficacia de dos modelos de detección aplicados a la Detección de Ataques de Denegación de Servicio DIS-flood. Éstos, serán entrenados para dar respuesta a dicha ofensiva. Entrenaremos y testearemos, pues, los siguientes algoritmos: Maquinas de Vector de Soporte y Random Forest. Finalmente, tomaremos una decisión en función de la virtud de diferntes métricas; el modelo elegido pasará a propuesta para ser implemetnado en le IDS.

7. Breve descripción de la estructura de la memoria

El Capítulo III establece las bases de la inteligencia artificial, centrándose en el Aprendizaje Automático (ML) y el Aprendizaje Profundo (DL), así como los puntos clave de los Sistemas de Detección de redes (IDS). El Capítulo IV desarrolla la fase de simulación del ataque donde se presenta el escenario de operaciones y la configuración del IDS; se introduce como la parte más importante de la memoria. El Capítulo V aborda el análisis de los resultados de la fase de simulación, destacando la evolución de los mismos y del IDS. El Capítulo VI concreta las aportaciones al campo de estudio y expone en rasgos generales el coste del proyecto. El Capítulo VII constituye el último capítulo de la memoria en donde se discuten las líneas futuras de trabajo y se reflexiona sobre la aportación y limitaciones del mismo.

Capítulo II. Fundamentos de IoT y Ataques de Denegación de Servicio

Originalmente, Internet se definía como la red mundial computadoras que ofrecía el intercambio de información proporcionando servicios a los usuarios. En respuesta a la rápida expansión de la red de Internet surge el Internet de las Cosas (IoT, por sus siglas en inglés) que se presenta como un escenario de expansión que proporciona la comunicación de todo tipo dispositivos a través de

la Internet con el fin de abrirle campo a todos los ámbitos de nuestras vidas. Como un todo, IoT se sirve de dispositivos inteligentes y sensores cuya capacidad de procesamiento, almacenamiento y respuesta, destaca respecto a la red conocida hasta ahora. Los primeros vestigios de IoT se remontan a la década de los ochenta con la conexión de una máquina expendedora a internet inauguró un nuevo campo que se vería reforzado en el siglo XXI. Antes del mismo, en la década de los noventa, sonaba el nombre de Internet de las Cosas en diferentes artículos y ámbitos académicos. A partir de ese punto, IoT se relacionaría con sensores insertados a cualquier dispositivo, máquina u objeto que los soportara.

Dicho auge lo puede observar usted en el aumento de dispositivos inteligentes que usa en su hogar. Una de las principales áreas de participación es el hogar inteligente: electrodomésticos, dispositivos de iluminación y seguridad, entre otros, cuentan con conectividad a Internet. También lo puede observar en el transporte: sensores, cámaras, semáforos, incluso su propio vehículo; portan nuevos servicios para asegurar un desplazamiento más eficiente y seguro. Así, observamos en IoT una tendencia trasversal que suaviza las dificultades de varios dominios: industria, geolocalización, ciudades, agricultura, medioambiente, gestión, procesos, sostenibilidad, energía, transporte, tráfico, salud, seguridad, educación, turismo, comercio, empresa, entretenimiento, hogar, etc.

En la Figura 1, se muestra la evolución de la integración de dispositivos en redes IoT. Como se puede observar, hay una fuerte tendencia al incremento del uso de dicha red. Hasta el año 2030, se prevé un aumento de alrededor de 30 billones dispositivos inteligentes en todo el mundo.

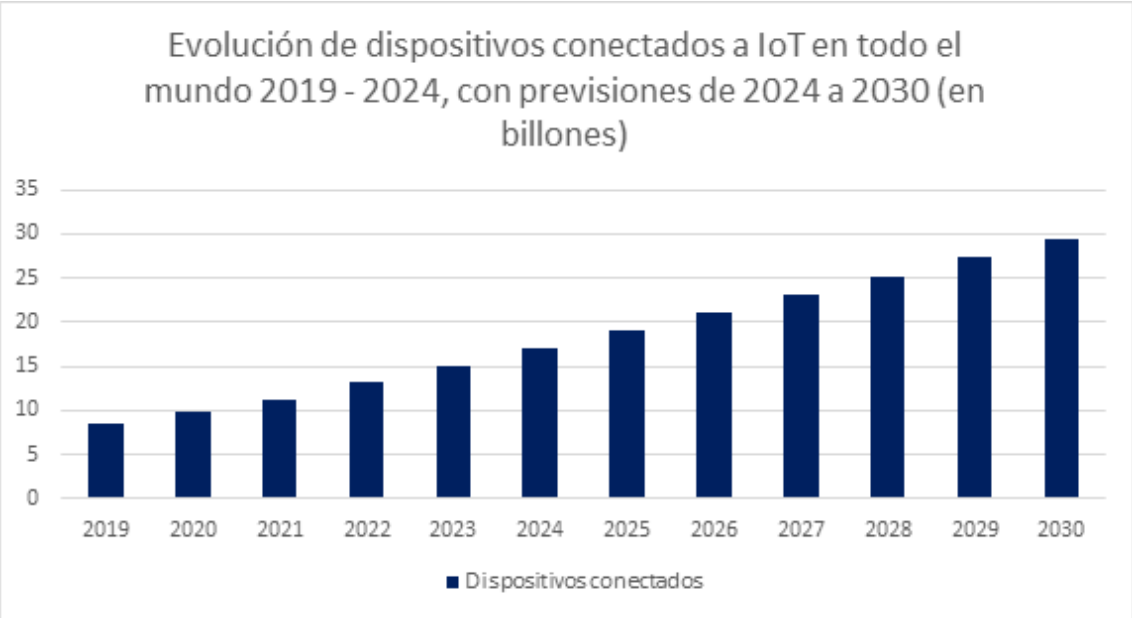


Figura 1: Evolución del uso de dispositivos conectados a IoT. Años 2019 - 2024, con previsiones de 2024 a 2030 (en billones) Fuente: [Webgrafía 6].

A pesar de ello, IoT todavía se encuentra en un punto poco maduro por lo que respecta a su seguridad y privacidad de datos, por lo que es necesario establecer un marco genérico de protección que contenga las cinco capas que componen la arquitectura IoT, estas son: la capa de percepción, la capa de red, la capa de procesamiento, la capa de aplicación y la capa de negocio. Empezando desde abajo, la capa de percepción engloba los elementos físicos, como chips, etiquetas o sensores, su homólogo en la capa OSI sería —si cabe— la Capa Física. Su función principal es recopilar datos e información para posteriormente enviarla a la capa de transporte. Dicha capa enlaza dispositivos inteligentes con la Capa de Procesamiento la cual procesará la información; servicios como *cloud* y bases de datos son las herramientas que introducen esta capa como la más importante de la arquitectura.

Figura 2: Arquitectura IoT

La Capa de Transporte, por su parte, conecta la Capa de Percepción y Procesamiento, es decir, opera a nivel de red mediante tecnologías como IEE 802.11, IEE 802.15, 3G o 4G. En un nivel superior encontramos la Capa de Aplicación. Ésta ofrece los servicios propios de aplicación: doméstica, optimización, logística, etc. Finalmente, la capa de negocio que gestiona el sistema. Controla la autenticación, estadísticas y servicios visualizando la información para planificar estrategias futuras adaptándose a los escenarios según la necesidad. La Figura 2 ilustra las capas que componen la arquitectura descrita.

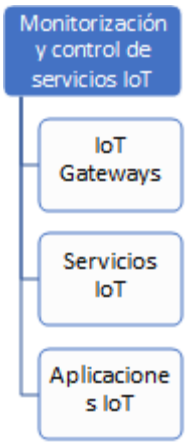


Figura 3: Estructura general de trabajo de IoT

Una vez introducida la arquitectura, todavía es conveniente una estructura genérica de operaciones. La Figura 3 muestra en sus variables de aplicación con conectividad entre servidores y dispositivos. El primer reto con el que nos encontramos a la hora de diseñar una arquitectura IoT es la escalabilidad e interoperabilidad. Por una parte, debe poder incorporar nuevos dispositivos al sistema, así como configurar un crecimiento continuo, manteniendo las condiciones de calidad. Mientras que, por otra parte, su diseño debe permitir las operaciones en distintos campos y funcionalidades. Por ende, un esquema general de red pertinente con los requisitos anteriores se compone en la Figura 4.

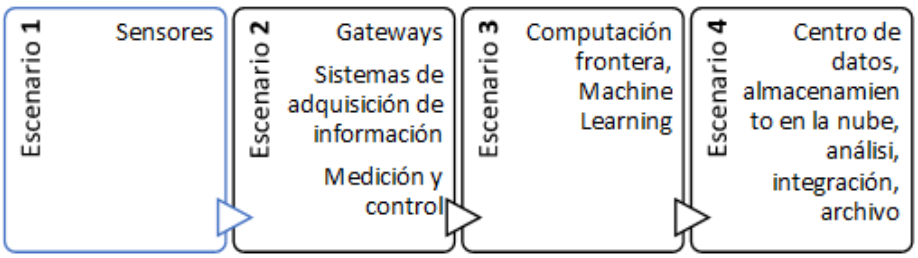


Figura 4: Escenarios de IoT

En el primer escenario, los sensores recopilan los datos —temperatura, presión, luz, movimiento—, y señales del mundo real y los transforman en información para ser analizada. El escenario 2 constituye un proceso de organización de los datos recopilados en que los sistemas de adquisición de información colaboran junto con los *gateway* agregando y optimizando una estructura o *stack* que va a ser transmitido al escenario 3. En este punto, la computación frontera maneja disposición de datos anterior proporcionándole diferentes funcionalidades, entre ellas, destacamos la visualización y el análisis mediante aprendizaje automático —éste último, desarrollará mejores enfoques de predicción—. Por lo que respecta al último escenario, éste procesa detalladamente y almacena las recopilaciones en un centro de datos, habitualmente, en un servicio de almacenamiento en la nube.

La integración de IoT en diferentes áreas de nuestras vidas abre la puerta a problemáticas tanto para los usuarios como los desarrolladores. Dos de esas problematicas es la seguridad y prevacidad. Estudiemos a continuación dichos campos.

1. Privacidad y seguridad

Como se ha mencionado anteriormente, la seguridad se ve comprometida ante vulnerabilidades, riesgos y ataques que efectúan los ciber-atacantes; más aún si se utilizan medidas pobres de autenticación, *software* desactualizado y encriptación ineficiente. Se debe presentar, pues, un enfoque de seguridad en todas las capas de la arquitectura. Prestando atención a las capas de transporte y aplicación, se han desarrollado protocolos de seguridad como el Protocolo de Seguridad de Capa de Protección (SSL) que se emplea como solución

criptográfica. No obstante, las medidas de protección dependerán de la tecnología, el estándar Wi-Fi, por ejemplo, supone un riesgo de seguridad importante.

En adición, la privacidad resulta ser el seguro del usuario. En IoT encontramos la inexactitud en cuanto a políticas de privacidad entre dispositivos inteligentes; cada elemento debería poder inspeccionar y reclamar aquellos requisitos que enmarquen el funcionamiento en una normativa sólida de prevención. Es en este escenario, donde encontramos otro aspecto importante: la interoperabilidad. La interoperabilidad se define como la función o la habilidad de conectar y transmitir datos entre dispositivos inteligentes. Encontramos cuatro niveles de interoperabilidad: semántico, sintáctico, técnico y organizacional. La enorme cantidad de elementos que participan en la red afinca un conjunto único y propio de sistemas con muy diferentes especificaciones y protocolos por lo que será necesario trabajar con enlaces de operación. En las redes modernas, se utilizan adaptadores y puertas de enlace que apaciguan esta demanda, no obstante, todavía se mantienen algunos desafíos.

La ética y la ley también devienen la esencia de los derechos regulatorios. Gran parte de las redes incorporadas apoyan regulaciones y normas con respecto a la protección de datos. Las principales normativas son:

- Reglamento General de Protección de Datos (RGPD).
- Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.
- Ley Orgánica de Protección de Datos (LOPD).
- Ley de Desarrollo de la Innovación y el Crecimiento de la Internet de las Cosas (DIGIT).

Por lo tanto, el objetivo debe ajustarse en el marco de dichas especificaciones.

2. Fundamentos de Ataques de Denegación de Servicio

El ataque más común y conocido en Internet es el Ataque de Denegación de Servicio (DoS). En esencia, consiste en copar o sobrecargar una red, servidor o sitio web mediante tráfico malintencionado; el objetivo, denegar los servicios e impedir que el tráfico de información fluya con normalidad. Dichos entorpecimientos se pueden lanzan en una gran variedad de entramados: redes de computadores, empresas, oficinas, escuelas, hospitales y redes IoT, entre otros. DoS, cuenta con la variante DDoS o Ataques de Denegación de Servicio Distribuido. Se trata de una modalidad de ataque que usa múltiples equipos —máquinas distribuidas que desempeñan el papel de *bots*— ejecutando el flujo indeseado de forma remota. Los *bots* operan de forma conjunta para ser transmitidos de forma masiva a espacios de interés. Así, pues, la principal diferencia entre ataques es la distribución. En la Tabla 1 se especifican a muy grandes rasgos las diferencias significativas.

Ataques DoS	Ataques DDoS
No distribuidos	Distribuidos
Impacto de calidad baja	Impacto de calidad severa
Relativa lentitud	Proceso inmediato
Origen único	Origen múltiple

Tabla 1: Diferencias generales entre DoS y DDoS

Como se puede apreciar, DDoS destaca sobre su antecesor DoS en cuanto a impacto y velocidad, esto se debe, razonablemente, por el efecto de la distribución. Estas ofensivas se pueden lanzar por una variedad típica de motivos, veamos algunos a continuación:

- Pentesting. Los desarrolladores y especialistas en ciberseguridad elaboran pruebas de penetración en entornos controlados con la finalidad de cubrir las posibles vulnerabilidades encontradas.

- Extorsión. Los delincuentes cibernéticos invaden la red de aquellos enemigos, gobiernos o entidades indeseadas por los mismos.

- Guerra cibernética. En situación de conflicto —guerra, crisis, conflicto bélico, etc—, los detractores interrumpen los servicios a modo de ofensiva para lograr datos o debilitar al enemigo.

- Entretenimiento o afición. Cibercriminales explotan las vulnerabilidades a modo de entretenimiento, prueba, testeo o afición.

3. Trayectoria DoS y DDoS

Los primeros ataques se comprenden entre la década de los setenta y noventa, años en que se elaboraron muchas pruebas a modo de experimento sobre terminales en ámbitos controlados de empresas y universidades. En los años noventa, con el auge de tecnologías de chat, como el protocolo *Internet Relay Chat*, surgieron los primeros ataques conocidos con intenciones maliciosas.

En 1996 fecha la primera ofensiva conocida a gran escala. El proveedor de servicios de internet, Panix, sufrió un tipo de ataque denominado SYN *flood* —inundación de mensajes de sincronización—. El hacker abordó la red de dichos mensajes obligando a cerrar los servicios por un periodo de 36 horas. En la Figura 5 se puede apreciar la dinámica de la ofensiva.

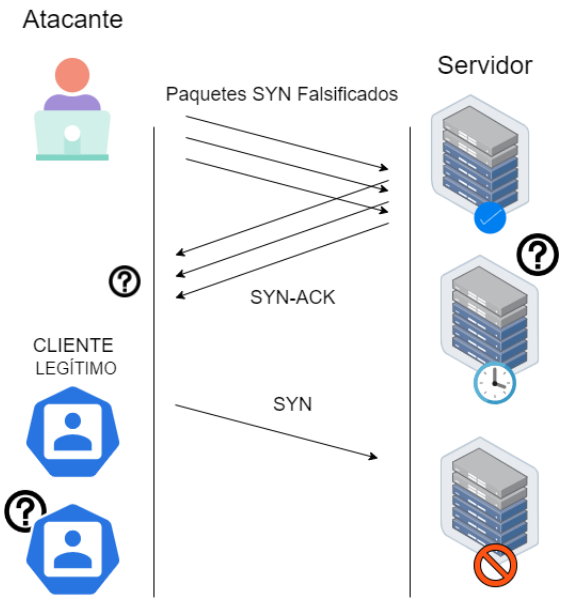


Figura 5: Ataque SYN flood

En este punto, se origina la evolución de los ataques de Denegación de Servicio. Numerosos dispositivos IoT fueron objetivos del secuestro y conversión de *bots* para el tráfico indeseado; se pasó rápidamente de gigabytes a terabytes. Por otra parte, el desarrollo de redes como 5G han permitido realizar intrusiones todavía más veloces y efectivas al mismo tiempo que los autores de las mismas pasen inadvertidos.

En la Figura 6 se pueden apreciar los principales países de origen de los ataques DDoS HTTP.

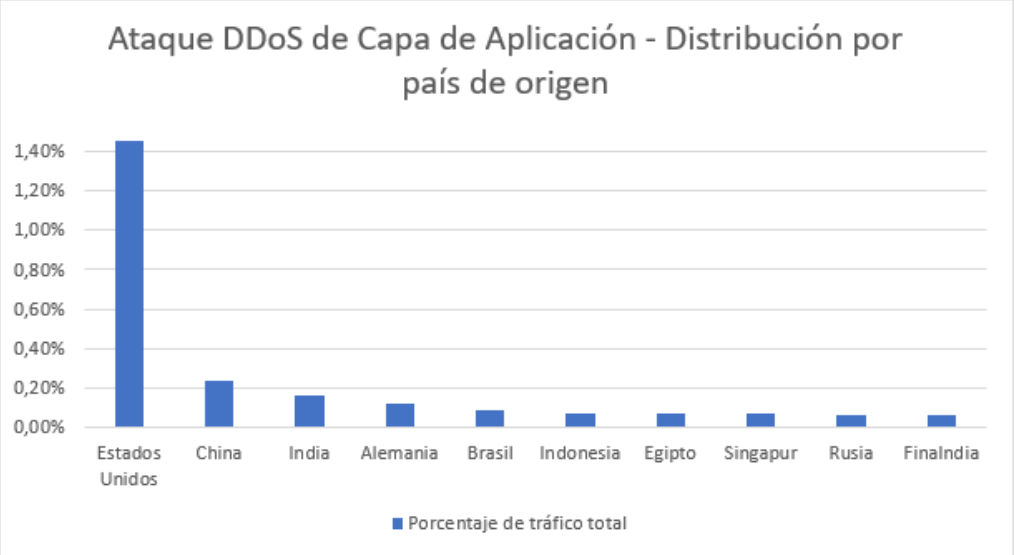


Figura 6: Ataque DDoS de Capa de Aplicación, distribución por país. Fuente: [Webgrafía 7].

Como se puede apreciar, Estados Unidos fue el país con más lanzamientos de ataques DDoS en HTTP; China se sitúa en segundo lugar. De ahora en adelante, y a menos que se indique lo contrario, todo lo descrito será referente a DDoS, dado que es más común, eficaz y rápido que su antecesor DoS.

4. Tipos de ataques en Capa de Transporte

En gran medida, DDoS opera en contra de los principales protocolos de la capa de aplicación y transporte del modelo OSI y TCP/IP. Estos son, TCP Syn *flood*, UDP *flood*, ICMP *flood* y solicitudes HTTP. Los ataques se dividen en dos vertientes: aquellos enfocados a IP versión 4 e IP versión 6. Los ataques de la capa 4 —Capa de Transporte— se subdividen en:

4.1 Ataques de tipo inundación TCP

Se introduce como una ofensiva conveniente tanto para IPv4 como para IPv6. La dinámica se centra en abordar el proceso de *3-Way Handshake* propio del protocolo TCP. En el proceso de conexión entre dos máquinas, el cliente o *host* envía paquetes de sincronización —SYN, abreviado— al equipo que figura como servidor. Posteriormente, éste último envía un mensaje de reconocimiento —SYN ACK— a modo de garantía, dando a entender que ha recibido la petición del cliente. Finalmente, el mismo responde con un ACK finalizando el proceso de conexión. En este punto, los dispositivos están conectados. No obstante, existen diferentes vulnerabilidades como vimos en la Figura 5; el atacante se enmascara con una dirección IP robada o elaborada falsamente enviando al objetivo mensajes SYN. En consecuencia, el servidor envía el respectivo SYN-ACK, aunque éste no llegará a ninguna parte dado que las direcciones de origen eran falsas haciendo que se mantenga abierta la sesión de establecimiento. En este punto, se seguirán enviando peticiones indefinidamente copando así el flujo; el servicio queda interrumpido.

4.2 Ataques de tipo inundación UDP

La siguiente categoría de ofensivas, a diferencia de la anterior, no se aprovecha de las vulnerabilidades del proceso de establecimiento de conexión. Sin embargo, utiliza 2 recursos de este protocolo: CHARGEN y ECHO. Cuando un usuario intenta conectarse al servicio CHARGEN, éste devuelve una cadena de caracteres y, si uno intenta conectarse al servicio ECHO, se le devolverá lo enviado al nodo entrante. En cuando un atacante conecte el puerto 19, es decir, el puerto de CHARGEN, al puerto de su objetivo —el puerto 7—, se producirá una transmisión masiva de información.

4.3 Ataques de tipo inundación ICMP

El protocolo ICMP se define como una herramienta de diagnóstico de problemas de comunicación y nos aporta información como el alcance del destino, informe de estado, datos de hosts de red, entre otros. Para tener una referencia, cabe mencionar que su uso más específico reside en el comando *ping* que usa la dinámica de *echo*. Un cibercriminal puede enviar peticiones desde diferentes IPs manipuladas colapsando la víctima. En la siguiente Figura 7 vemos la dinámica descrita:

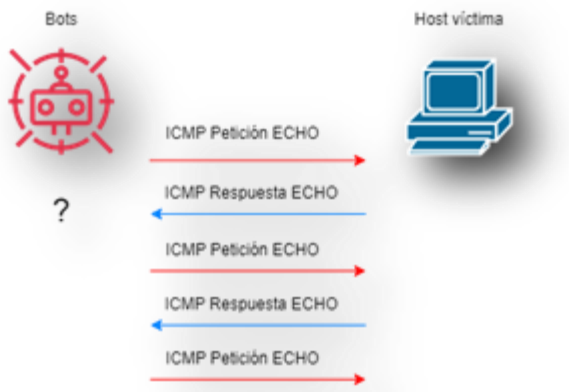


Figura 7: Ataque de inundación ICMP

5. Tipos de ataques en Capa de Aplicación

Analicemos a continuación los homólogos de la Capa 7 del modelo OSI: la Capa de Aplicación. Se subdividen en:

5.1 Aprovechamiento método GET de solicitudes HTTP

En la Capa de Aplicación se opera constantemente con protocolos HTTP para el intercambio dedicado a servidores de alojamiento de páginas web. El usuario malicioso puede enviar tramas HTTP con un encabezado incompleto de forma constante agotando los recursos disponibles lo cual se traduce en la invalidación de los servicios. Al ser un ataque tan efectivo, se puede desarrollar solamente con DoS sin la necesidad de usar otros equipos a modo de *bots*. En la Figura 8 se procede a una comparativa entre el funcionamiento normal de peticiones HTTP y aquel orientado a la denegación de servicio.

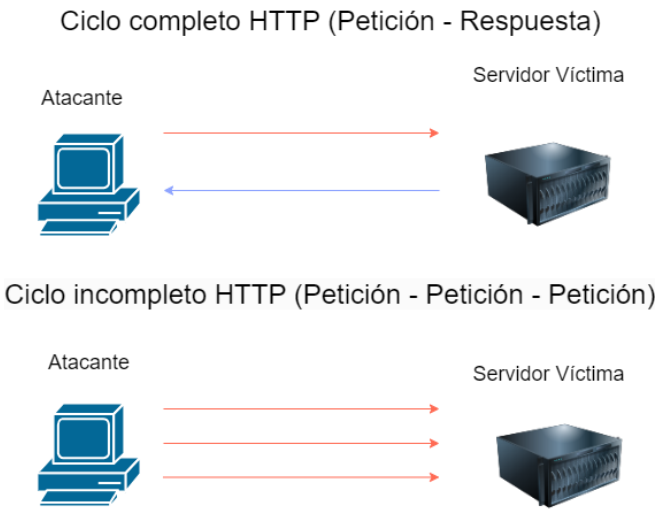


Figura 8: Comparativa ciclos HTTP ante ataque

5.2 Solicitudes incompletas de HTTP con POST

Este método de ataque tiene cierto parecido con el anterior. No obstante, se diferencia en el enfoque; mientras el anterior se aprovecha del método GET, éste usa el método POST. Se trata de una herramienta que introduce los parámetros en la solicitud HTTP presentándose como un método muy flexible. En la Tabla 2 se muestra una comparativa entre los dos métodos.

Método GET	Método POST
Para la configuración de sitios web	Para la transferencia de datos
Visible para el usuario	No visible para el usuario
Permite parámetros ASCII	Permite ASCII y binarios
Parámetros URL guardados junto a la misma	Parámetros URL no guardados

Tabla 2: Comparativa métodos GET y POST.

Como se puede apreciar, la principal diferencia radica en su utilidad, mientras es idóneo para la personalización de páginas web, el segundo es perfecto para cuando se pretende transmitir datos al servidor. Así pues, por similitud con el ataque anterior, no se extenderá la descripción más allá del punto I.

Discutamos a continuación las oportunidades de los ciberdelincuentes usando IPv6 en el proceso de detección de vecino.

5.3 Ataque de dirección duplicada

IPv6 nos permite configurar la dirección sin estado, es decir, permite la posibilidad de asignarse direcciones IP globales —no utilizadas— enviando dichas comprobaciones al host vecino en formato *multicast*. En este punto, entra el sistema de detección de dirección duplicada (DAD) el cual puede ser intervenido por el usuario maligno, pues éste puede anunciar su vecindad con respuestas falsificadas, negando de esta manera, la apropiación de una dirección IPv6 al primer host. En la Figura 9 se puede apreciar el funcionamiento del mecanismo descrito.

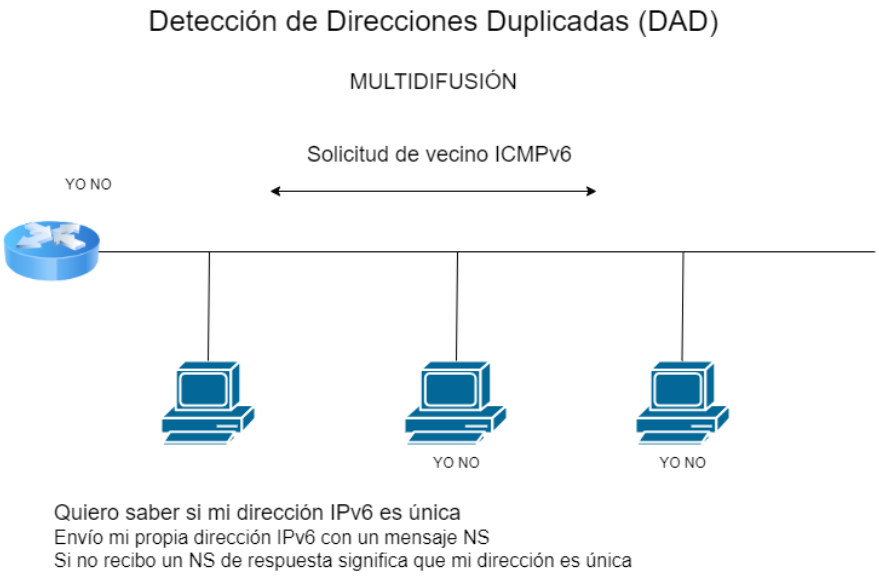


Figura 9: Funcionamiento mecanismo DAD.

5.4 Ofensiva a través del enrutador

Los *hosts* cuentan con tablas de direcciones IP de direcciones de *router* predefinidos. Los atacantes son capaces de elaborar mensajes falsos de anuncio de enrutador cuyo tiempo de vida es cero. En consecuencia, el *host* es engañado para que envíe los datos a los nodos a través del enrutador. No obstante, como no existe tal nodo, los paquetes no llegarán a un destino concreto y se sobrecargará la red.

6. Análisis de tendencias de ataque en capa de aplicación

Desde el año 2019 se ha visto una creciente tendencia de ataques DDoS. En la Figura se puede apreciar la distribución de métodos de ataque dedicados mediante protocolos de la capa de aplicación. Los datos se introducen trimestralmente para el periodo que comprende desde el primer trimestre de 2019 hasta el primer trimestre de 2021. Podemos ver que la ofensiva basada en

el protocolo HTTP es el más frecuente. No obstante, se produce una disminución del 10.70% en el paso del cuarto trimestre de 2019 —disminución del 20%— al primer trimestre de 2020 —disminución del 30%—. A partir del cuarto trimestre de 2019 hasta el primero del año 2021 se disminuye el uso de ataques basados en protocolos de aplicación. El pico más álgido se registró en el segundo trimestre de 2019: 25.30%, mientras que en el primer trimestre del año 2021 ascendió a 9.32%, es decir, una disminución del 15.97%.

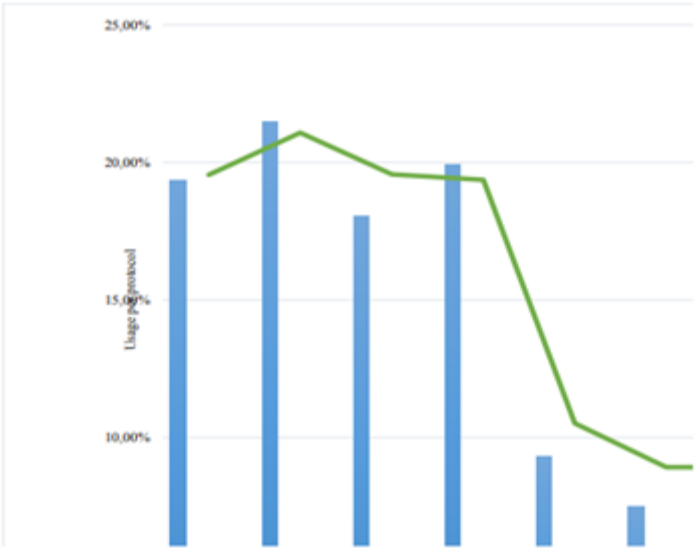


Figura 10: Frecuencia de uso de protocolos de capa de aplicación en la realización de ataques DDoS durante el período de Q1-2019 a Q1-2021 Fuente: [Webgrafía 20].

Capítulo III. Fundamentos de Inteligencia Artificial

La inteligencia artificial (IA) se define como un sistema informático diseñado mediante algoritmos, cuyo propósito es realizar operaciones que requieren de inteligencia humana. Es, por tanto, una disciplina plural y de uso extensivo en diferentes campos de la informática.

La encontramos en los diferentes dominios:

1. Teoremas

   Se trata de la aplicación más antigua de la inteligencia artificial en donde utiliza fundamentos de control y razonamiento para demostrar teoremas.

2. Procesamiento del lenguaje

Encontramos el diálogo hombre–máquina que desarrolla la generación e interpretación de oraciones, la traducción por computadora o la indexación automática, entre otros. Destaca, en consecuencia, las técnicas de razonamiento contextual.

3. Reconocimiento de imagen

Junto con la demostración de teoremas, el reconocimiento y procesamiento de patrones fue una de los primeros campos de la inteligencia artificial. Las aplicaciones de reconocimiento avanzadas abrazan el diagnóstico e inspección de imágenes, en consecuencia, destaca el razonamiento artificial dedicado a la visión.

4. Robótica

Sin duda uno de los campos scon mayor implicación en la inteligencia artificial es la robótica; máquinas capaces de realizar movimientos repetitivos aprendidos en fases preliminares. Si bien es cierto que en sus orígenes se ahondaba más en la labor mecánica, la nueva generación de robots incorpora métodos de IA como por ejemplo sistemas de visión, razonamiento, planificación dinámica de acciones, etcétera.

La inteligencia artificial se basa en el conocimiento que subyace a la experiencia humana, en consecuencia, resalta la importancia de la implementación de esquemas de razonamiento más complejos. Con ello, se consiguen dinámicas similares en que un ser humano razona. Veamos a continuación los esquemas de razonamiento más importantes.

El razonamiento aproximado es un esquema que opera con datos inexactos basándose en la aceptación de datos imperfectos mientras se produce un proceso lógico. La clave de este modelo se centra en el manejo de la imprecisión e incertidumbre. La lógica difusa, por tanto, se introduce como aquel razonamiento artificial con parecidos al razonamiento humano, pues involucra posibilidades intermedias entre la afirmación y la negación.

El razonamiento basado en modelos se basa en reglas que asocian hechos a conclusiones. Se trata de la unión de lo superficial con lo profundo, es decir, son reglas de producción heurística que confluyen en tres aspectos: experiencia, práctica y observación, —por ejemplo, separar lo dado de lo buscado—. Las arquitecturas basadas en esta categoría se dividen según su naturaleza como las redes semánticas o las redes causales, entre otras.

En cuanto al razonamiento basado en casos, introduce el paradigma propio de razonamiento concerniente a la analogía, esto lo vemos en la Figura 11. Dicho razonamiento en confiar en experiencias anteriores para resolver problemas presentes. Es pues, que un sistema práctico que necesita dar respuesta a la elección de casos, su representación e indexación.

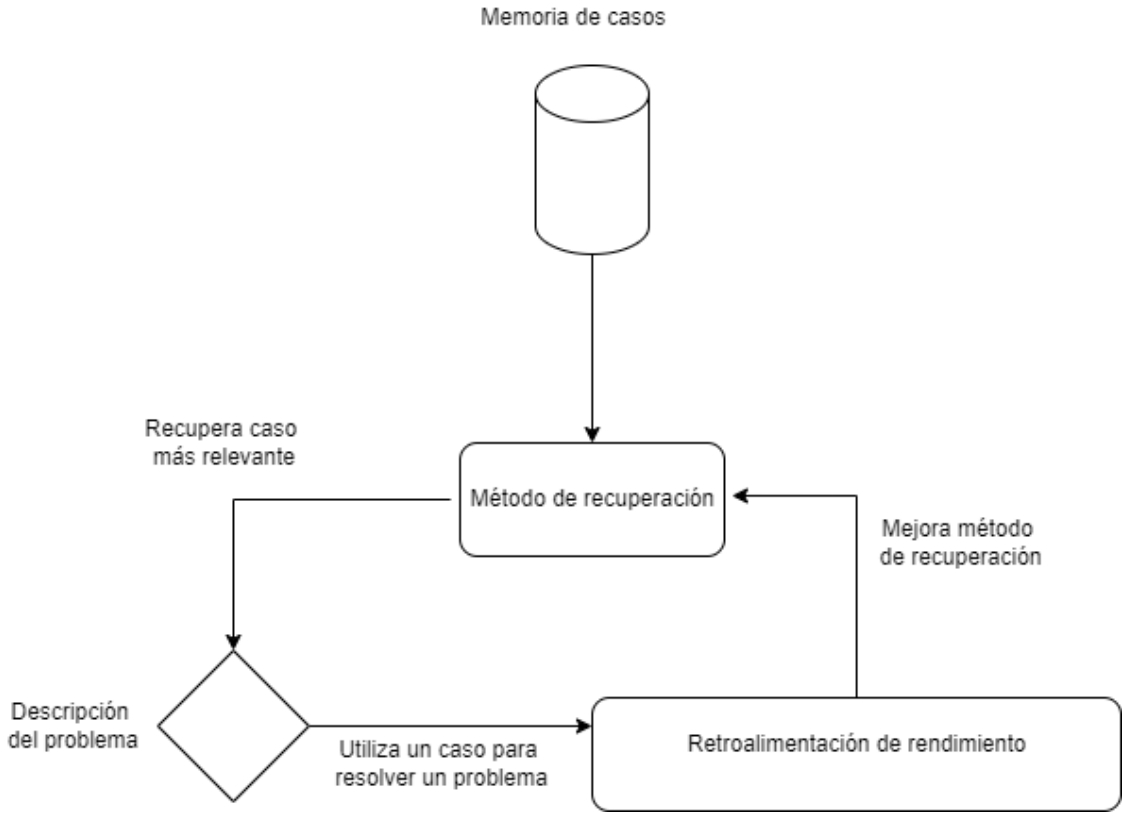


Figura 11: Principio de razonamiento basado en casos

En la Figura 11 anterior se puede apreciar el principio de razonamiento basado en casos. La memoria de casos almacena los casos anteriores para luego producirse la búsqueda del caso más relevante de la memoria. Seguidamente, se describe el problema específico que se está tratando de resolver, para, finalmente, retroalimentarse con el fin de ayudar a mejorar el método de recuperación.

Por último, los sistemas de razonamiento basado en restricciones también presentan un papel importante en la inteligencia artificial. Las restricciones no son más que declaraciones influidas sobre variables con naturalezas distintas dependiendo de la aplicación: simbólica, geométrica o numérica. Estos sistemas operan con algoritmos con el propósito de reducir el abanico de soluciones a investigar.

1. Aprendizaje Automático (ML) y Aprendizaje Profundo (DL)

Los sistemas inteligentes capacitados con inteligencia no humana se basan en el aprendizaje automático (ML). El aprendizaje de carácter automático acontece la capacidad o virtud de dichos sistemas para aprender a partir de datos de entrenamiento previamente establecidos para el problema a resolver. En otras palabras, es el proceso de automatización de la construcción de modelos analíticos que resuelven tareas asociadas. El aprendizaje profundo (DL), en contraposición, es aprendizaje automático cuya base de operación son las redes neuronales artificiales.

La formación automática busca comprender automáticamente patrones significativos a través de observaciones. A día de hoy, el ML se encuentra ante el escenario de surgimiento de sistemas capaces de realizar procesos cognitivos similares a los de los humanos. Tal capacidad se fundamenta en modelos analíticos que elaboran respuestas y predicciones. Las primeras tentativas de construcción de modelos analíticos se respaldaban en la programación dedicada de lógicas y procedimientos conocidos en sistemas inteligentes por medio de reglas diseñadas a mano.

La evolución de las redes neuronales artificiales hacia otras más profundas ha constituido el mayor avance en cuanto a algorítmicos sofisticados y técnicas de procesamiento. Tanto es así, que en determinados contextos el ML aporta signos de rendimiento elevado —sobrehumano— que supera las capacidades hasta ahora conocidas.

A continuación, se asentarán los fundamentos de ML y DL proporcionando una distinción conceptual entre significados explicando el proceso de construcción automatizada de modelos a través de los aprendizajes mencionados.

2. Diferencias entre ML y DL

Como ya sabemos, la inteligencia artificial permite a las computadoras reproducir comportamientos humanos para resolver tareas con nula o mínima intervención de estos. También se ha descrito anteriormente que en los orígenes de la IA se abordó un primer enfoque consistente en declarar mediante codificaciones en lenguaje formal. No obstante, este escenario ofrece ciertas acotaciones, puesto que los seres humanos se esfuerzan en explicar el conocimiento intuitivo, inconsciente y adquirido mediante experiencia para las tareas dificultosas.

Previamente a la distinción entre conceptos: a) algoritmos de aprendizaje automático, b) redes neuronales artificiales y c) redes neuronales profundas, es necesario contar con una base sólida sobre inteligencia artificial. En la Figura 12 se muestra la relación entre a), b) y c).

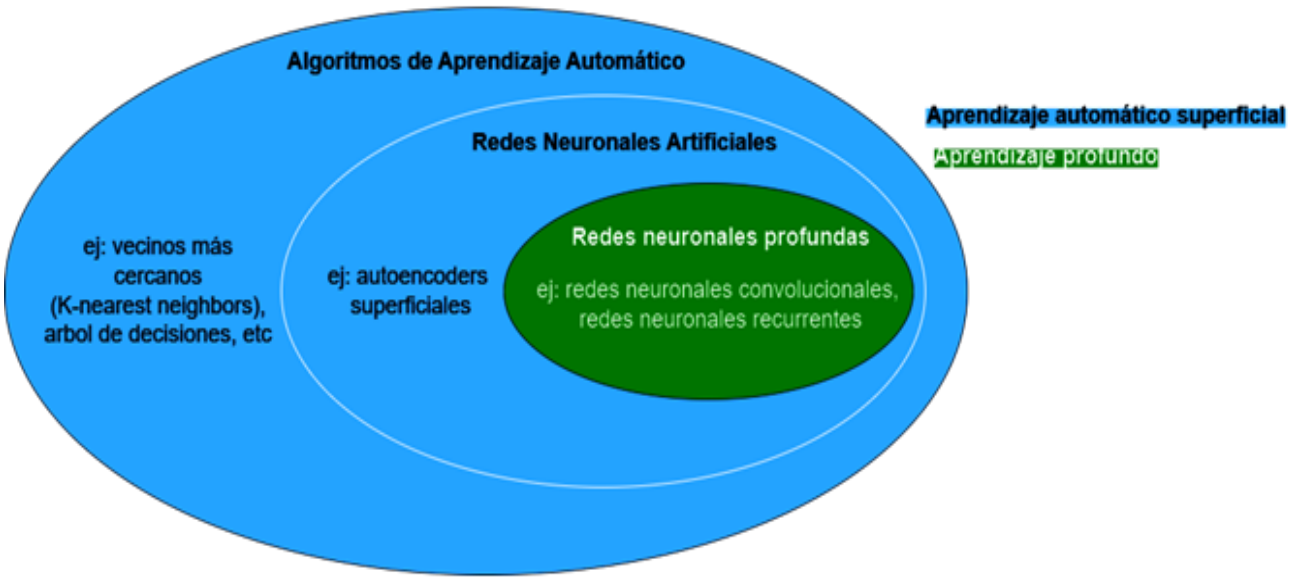


Figura 12: Conceptos y clases de algoritmos de aprendizaje automático

En función del campo de aplicación del aprendizaje, se ofrecen distintas clases de algoritmos de ML, siendo cada una de ellas múltiples variantes que incluyen modelos de regresión —analizar y predecir relaciones entre variables—, algoritmos basados en instancias —se comparan nuevas instancias de problemas con instancias de entrenamiento—, arboles de decisión —aprendizaje supervisado para clasificación y regresión—, métodos bayesianos —interpretación de la probabilidad como medida de confianza— y redes neuronales artificiales (ANNs).

Respecto a los tres primeros tipos de aprendizaje automático, las redes neuronales cuentan con una estructura que aporta flexibilidad ante modificaciones para un sinfín de contextos en los tres tipos anteriores. Las ANNs se fundamentan en el principio de procesamiento de información en sistemas biológicos, pues representan matemáticamente unidades de encausamiento conocidas como neuronas artificiales. La mejor analogía de lo descrito es el funcionamiento del cerebro humano; toda conexión entre neuronas envía señales que pueden ser modificadas —amplificadas o atenuadas— por un peso que se ajusta de manera continua durante el aprendizaje. En palabras más sencillas: en el aprendizaje automático, estos pesos se ajustan durante el entrenamiento; cuando el modelo comete errores, aprende y ajusta los pesos para mejorar sus predicciones. Comúnmente, las neuronas se organizan en redes que a su vez se organizan en capas; una capa de entrada (*in*) toma la entrada de datos y la capa de salida (*out*) realiza el efecto final. Por ejemplo, se obtienen imágenes de figuras (*in*) y se clasifican según su forma (*out*). Entre ese proceso se pueden formar otras capas adicionales encargadas de aprender un mapeo no lineal entre *in* y *out*. Sin embargo, en este punto cabe destacar que los parámetros como número de capas, neuronas, funciones de activación o tasas de aprendizaje deben ser

configurados manualmente porque no pueden ser aprendidas por el propio algoritmo.

En cuanto a las redes neuronales profundas, se organizan en más de una capa adicional implementadas en arquitecturas profundamente anidadas. En adición, sus neuronas introducen mejores virtudes al ser avanzadas respecto a las ANNs de carácter simple: múltiples activaciones mononeuronales, convoluciones... Estas particularidades hacen que las redes neuronales profundas puedan recibir datos sin procesar y concluyen de forma automática representaciones necesarias para la tarea de aprendizaje correspondiente. En esencia, esta es la característica principal de las redes, que se conoce como aprendizaje profundo.

A diferencia de lo anterior, las ANNs simples y otros algoritmos pueden ser clasificados como "superficiales" y comprensibles por los humanos, esto se considera visto como cajas blancas. Sin embargo, la mayoría de algoritmos de aprendizaje profundo son vistos como indescifrables; cajas negras. Una de las principales diferencias entre el aprendizaje profundo y el superficial es que mientras el primero es útil en dominios de datos grandes y de alta dimensión, el segundo se desenvuelve bien en contextos de disponibilidad limitada de datos de entrenamiento.

Antes de pasar a las siguientes secciones, es necesario diferenciar los tipos de aprendizaje automático. Se describen a continuación en la Tabla 3 como: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

Tipo	Descripción
Aprendizaje supervisado	Se trata de un conocimiento que exige datos de entrenamiento que cubra ejemplos para la entrada, tanto como respuestas etiquetadas con el fin de entrenar algoritmos que prevén datos de forma precisa. En función de los datos introducidos en el modelo, la formación adapta sus ponderaciones hasta que el modelo se ajuste correctamente, esto, ocurriendo en el proceso de validación cruzada. Este modelo utiliza datos dedicados para el entrenamiento que le enseñan a lanzar la respuesta deseada. Dichos datos incluyen entradas y salidas correctas que permiten al algoritmo a medir su exactitud ajustándose hasta reducir el error considerablemente. La presente formación se divide en dos categorías de problemas: • Problemas de regresión. Se utiliza para distinguir entre variables dependientes e independientes. Fundamentalmente se emplea para predecir un número. • Problemas de clasificación.

	Mediante algoritmos adjunta de forma precisa datos de prueba a categorías concretas. Identifica entidades particulares en el conjunto de datos y procura extraer inferencias sobre cómo deberían ser etiquetadas o definidas dichas entidades. Entre los algoritmos de clasificación más frecuentes se incluyen los clasificadores lineales, las Máquinas de Vectores de Soporte (SVM), los árboles de decisión y el algoritmo de los K vecinos más cercanos.
Aprendizaje no supervisado	El aprendizaje no supervisado hace uso de procesos para clasificar en clústeres agrupaciones de datos, a diferencia de lo anterior, sin etiqueta. Son capaces de, sin intervención humana, operar en solitario para analizar patrones ocultos. Es ideal para el estudio y análisis de datos exploratorios, reconocimiento de imágenes, etcétera, debido a que son muy buenos en concluir símiles y diferencias en la información. Esta cuestión trabaja en tres áreas principales que son: agrupación de clústeres, asociación de dimensionalidad y su reducción. • Agrupación de clústeres. Esta técnica de minería de datos agrupa informaciones no etiquetadas según parecidos o diferencias. Su principal función radica en procesar objetos de datos sin clasificar y sin formar parte de patrones o estructuras de información. • Asociación de dimensionalidad. Esta regla se fundamenta en una metodología de que detecta aspectos comunes entre variables de un conjunto de datos. El mejor ejemplo para entender esto son los análisis que nos ofrecen ciertas páginas o plataformas, como la recomendación de productos y compras que ya realizaron otros clientes —véase la cesta de la compra de tiendas *online*—. • Reducción de dimensionalidad. Contrariamente a lo que se puede pensar, una gran cantidad de datos para conseguir resultados más precisos no siempre es conveniente pues se puede dar una situación de sobreajuste y, en consecuencia, entorpecer la visualización del conjunto de informaciones. La técnica de reducción de dimensionalidad se utiliza para este propósito; reducir la entrada de datos a cantidades de fácil gestión.
Aprendizaje por refuerzo	El aprendizaje por refuerzo dista un poco de los puntos anteriores. Este sistema no opera con entradas y salidas, sino

	que se producen dinámicas de descripción del estado del sistema, seguidamente se elabora un objetivo —con sus respectivas acciones y restricciones—, y exponemos el algoritmo al principio de prueba y error con el fin de lograr el objetivo con la máxima retribución. El ámbito más común de aplicación son los videojuegos.

Tabla 3: Tipos de aprendizaje automático.

3. Sistemas de Detección de Red (IDS)

Para hacer frente a las intrusiones de los sistemas, los desarrolladores presentan diferentes herramientas de detección tanto a nivel de red y a nivel de *host*. Por una parte, los Sistemas de Detección de Red —IDS, por sus siglas en inglés— se encargan de controlar, a través de dispositivos y *gateways,* el flujo anormal de red. En segundo lugar, los sistemas a nivel de *host* observan las operaciones de equipos, como sistemas informáticos o nodos de red.

Los Sistemas de Detección de Intrusos a Nivel de Red (NIDS) se agrupan, según el enfoque de detección, en tres categorías principales:

I. IDS basados en firmas

   Éstos se basan en firmas, reglas o dinámicas de ataque anteriores guardadas en bases de datos; es decir, se trata de una comparación de incidencias. Su efectividad es total para ataques conocidos, pero es vulnerable cuando se trata de ofensivas desconocidas.

II. IDS basados en anomalías

   Su objetivo es comparar el tráfico de red mediante técnicas estadísticas con otros considerados normales para detectar ataques. No obstante, presenta problemáticas en cuanto a la precisión cuando se producen falsos positivos.

III. IDS basados en modelos híbridos

   Se trata de la combinación de los modelos A y C que aprovecha las virtudes de ambos.

4. Algoritmos típicos de detección

Uno de los marcos más usados de protección es la mitigación mediante *Fog Computing*. Este marco utiliza enfoques de mitigación basados en anomalías que proporciona una detección rápida, todo empleando aprendizaje automático.

En la Figura 13 se muestra el flujo de operaciones. En primer lugar, se analiza el tráfico de red para ser comparado con dinámicas anteriormente almacenadas en la base de datos: en este punto se abren dos nuevos escenarios; si se detecta un ataque, se procede al bloqueo del flujo y el administrador de la red recibe una notificación de incidencia, sino se procede a la clasificación del flujo. La función del clasificador radica en concluir la normalidad del tráfico comparándolo con un flujo de categoría normal anteriormente creado. Si se concluye una anormalidad en el flujo, se actualizará la firma en la base de datos para posteriores comparaciones y se bloqueará el tráfico.

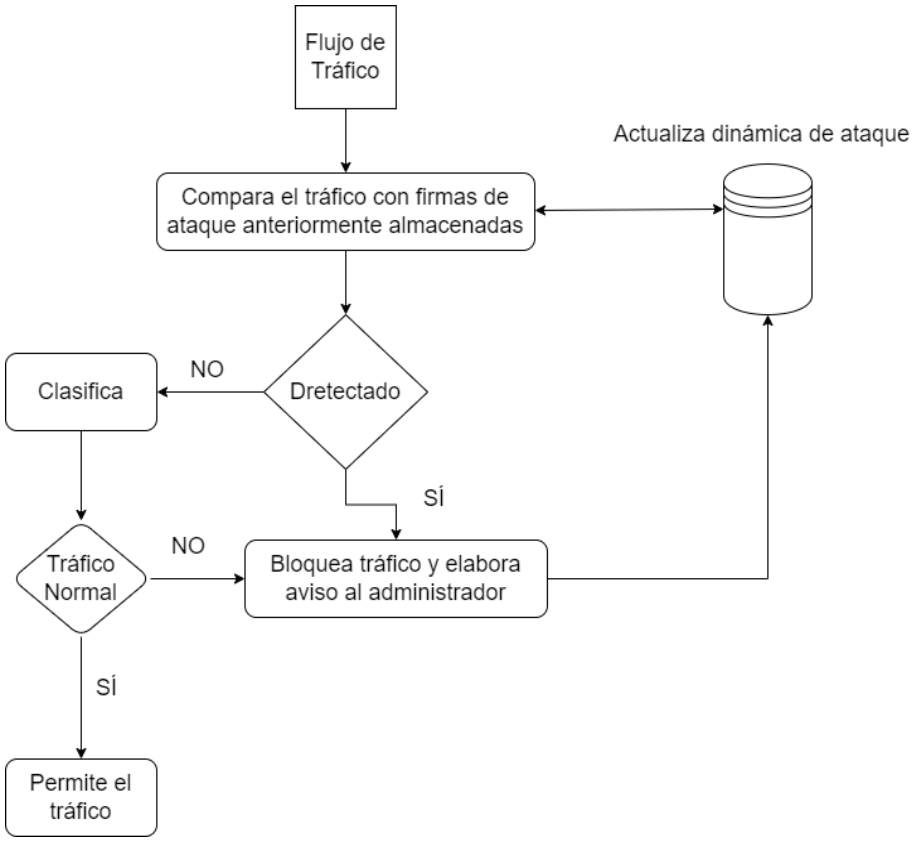


Figura 13: Flujo de operaciones basado en Fog Computing

El algoritmo que utiliza el clasificador para tal propósito es el *k-Nearest Neighbors* o también conocido como k-NN. Se trata de un proceso de aprendizaje automático supervisado —es decir, su proceso de aprendizaje utiliza datos previamente categorizados— típicamente utilizado en redes de computadores e inalámbricas que clasifica la información desconocida basándose en información conocida, es decir, el algoritmo se basa en que los datos desconocidos son similares a aquellos más cercanos en un espacio n-dimensional.

Primeramente, define el número de vecinos (k) al dato desconocido, que será utilizado para la clasificación. Seguidamente, se calcula la distancia (d) a todos los puntos de datos conocidos en el espacio de entrenamiento. Para ello,

calculamos la distancia entre dos puntos —el conocido y el desconocido— mediante la fórmula de distancia euclidiana:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figura 14: Distancia euclidiana entre dos puntos Fuente: [Webgrafía 8]

Finalmente, se clasifica la naturaleza del dato desconocido en función del grupo más cercano y común de los k vecinos. En la Figura 15 se analiza singularmente el funcionamiento de algoritmo:

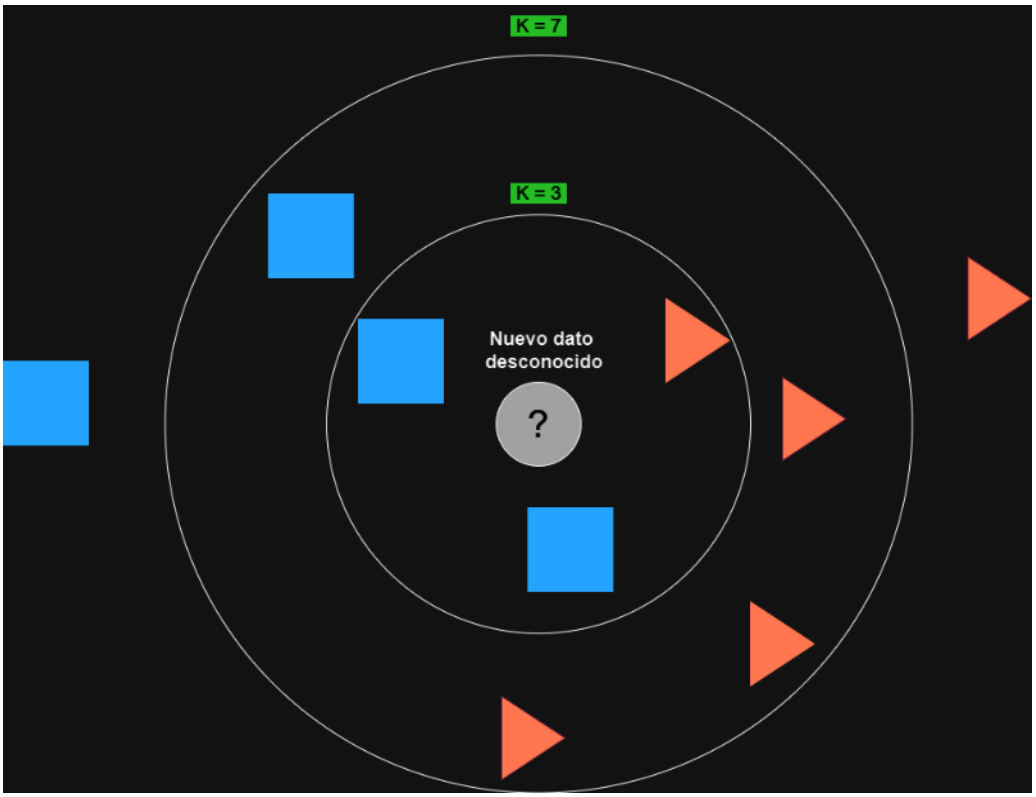


Figura 15: Clasificación básica según el protocolo k-Nearest Neighbors

En el caso de K = 3, el dato desconocido se clasificará como un cuadrado, porque k engloba más cuadrados (2) que triángulos (1). Por otro lado, en caso de que K = 7, el dato desconocido se clasificará como triángulo, porque k engloba más triángulos (4) que cuadrados (3).

Otro marco de protección común es el de Redes Definidas por Software (SDN). Se trata de una arquitectura que permite el cumplimiento de dos procesos: separación del plano de control y de red. En arquitecturas de red normales, las denegaciones de servicio se detectan basándose en características del tráfico y anomalías del mismo. Mientras el primero recopila las características de

ofensivas anteriores en una base de datos para luego ser comparadas con el flujo de red, el segundo establece un modelo propio y lo compara.

La arquitectura SDN ofrece control a través de la visión de toda la red, su implementación es flexible y su definición por *software* consolida una respuesta rápida de las políticas de tráfico. El algoritmo de mapas auto-organizados (SOM, *Self-Organizing Maps*), se presenta en este marco como un recurso de detección de DDoS sustrayendo las estadísticas del flujo del mismo. Este método tiene beneficios en cuanto al consumo y detección. Contrariamente al algoritmo k-NN, es una técnica de aprendizaje no supervisado —no conoce, en primera instancia, valores categóricos ni numéricos— y establece que los datos de entrada se organizan en un plano bidimensional en que los puntos cercanos comparten patrones similares en los datos iniciales. Véase en la Figura 16 el funcionamiento del proceso descrito:

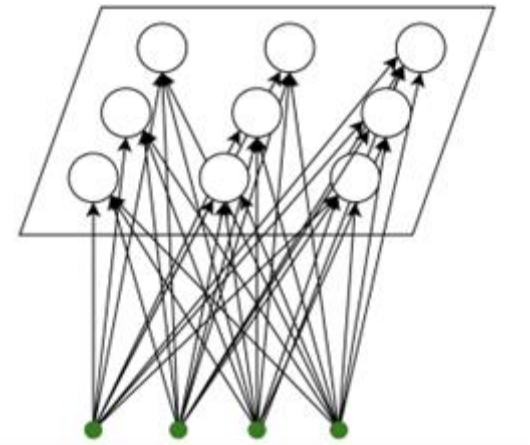


Figura 16: Reducción dimensional en SOM. Fuente: [Webgrafía 9].

Se produce lo siguiente:

1. Cada nodo se inicializa.
2. Se escoge un vector al azar.
3. Se determina qué nodo tiene los pesos más parecidos al vector de entrada (BMU, Mejor Unidad de Coincidencia en inglés).
4. Se calcula los alrededores del BMU.
5. El peso ganador es recompensado con atributos parecidos al vector de muestra. También se recompensa a los vecinos.
6. En función de la cercanía de los nodos al BMU se aumentan o disminuyen los pesos.
7. Se repite el paso 2 N veces.

A diferencia de los algoritmos anteriores, en este escenario no se necesita la etiqueta de ataques para detectarlos además de ser adaptables al cabio de tráfico y visualización para detectar incidentes. No obstante, debemos tener en cuenta que su coste computacional es elevado y presta a diagnosticar falsos positivos, pues las desviaciones de tráfico no siempre son sinónimo de ataque.

3. Extracción de características

La construcción de modelos se fabrica a partir de la extracción de características de grandes conjuntos de datos. En este contexto, se define característica como la descripción de un dominio o propiedad perteneciente a la entrada de datos para realizar una representación correspondiente. De este modo, este primer paso consiste en obtener características preservando información discriminatoria para la función de aprendizaje. Un buen ejemplo de ello que podemos ver en nuestro día a día es la agrupación por utilidad de las reseñas de los clientes en un comercio en línea. Es clave, pues, un proceso exitoso de obtención de características porque el resultado exitoso depende de ello.

De forma general, dicho desarrollo procede de manera jerárquica, donde las características de carácter abstracto de alto nivel se ensamblan partiendo otras más simples.

4. Construcción del Modelo

La construcción del modelo analítico depende en gran medida del tipo de algoritmo de aprendizaje. Veamos a continuación dos casos comunes.

En la construcción de modelos de clasificación, son los algoritmos de árbol (*tree algorithms*) de decisión los que operan en el espacio de características partiendo de forma incremental los datos en subdivisiones cada vez más homogéneas, todo, siguiendo una estructura de árbol. Por otra parte, las Máquinas de Vectores de Soporte buscan construir un hiperplano discriminatorio entre puntos de datos; las entradas se proyectan en un espacio de características de mayor dimensión.

En contraposición a lo anterior, el aprendizaje profundo (DL) construye modelos directamente con datos de entrada de dimensión elevada, es decir, aprovecha su virtud de instrucción automática de características. En consecuencia, vemos arquitecturas DL organizadas de extremo a extremo que combinan ambos puntos en un único *pipeline.*

Dependiendo de la utilidad que le vamos a dar, utilizaremos una u otra arquitectura. La Tabla 4 nos muestra una descripción de las principales arquitecturas de *deep learning*.

Arquitectura	Descripción
Representación Distribuida.	Las representaciones distribuidas son cruciales en dos factores: el aprendizaje de características y modelado del lenguaje en procesamiento de lenguaje natural (NLP). Una vez descrito el primer punto, nos disponemos a describir el segundo: en NLP propiedades del lenguaje tales como las palabras o las oraciones se proyectan de forma numérica en un espacio semántico unido en un formato *embedding* (incrustación). Las incrustaciones de palabras codifican palabras en vectores de características densas con baja dimensionalidad. En definitiva, este proceso respeta las relaciones semánticas entre palabras superando el problema de las codificaciones dispersas. En términos más sencillos: las palabras que se dan en contextos similares están posicionadas cerca de otras en el espacio vectorial. Esto es lo que utilizan las inteligencias artificiales capaces de responder preguntas. Comúnmente, la representación distribuida se implementa con redes neuronales recurrentes (RNNs) para realizar distintas tareas.
Red Neuronal Recurrente (RNNs)	Se trata de una arquitectura de red neuronal diseñadas para agrupaciones de información secuenciales —secuencia de sucesos, secuencias reprimidas por el tiempo, etcétera—. Se basa en la retroalimentación gracias a la operación de bucles, por lo que aprende patrones secuenciales modelando conclusiones temporales. Esta tecnología se divide en dos niveles: RNN simples y RNN complejas. Se diferencian en que mientras las primeras ven mermada su memoria temprana, las segundas presentan una solución a este problema.
Red Neuronal Generativa Adversaria (GAN)	La esencia de esta red neuronal radica en formarse en la distribución de probabilidad sobre un grupo de datos de entrenamiento con el fin de que la red pueda elaborar —de forma aleatoria—, nuevas muestras de datos. A consecuencia de lo anterior, separamos GAN en dos subredes que compiten entre ellas. Por una parte, encontramos la red encargada de captar la distribución de entrada y generar ejemplos nuevos. Por otra parte, encontramos una red de carácter discriminatorio que pretende elaborar distinciones entre ejemplos reales y artificiales. Como hemos dicho, compiten entre ellas, por lo que, cuando una gana en el entrenamiento, la otra pierde, entonces seguirán así hasta que la subred discriminadora sea incapaz de distinguir entre real y artificial. Entre sus usos, lo podemos ver en el diseño de arte o en transiciones de representación, como peticiones de pasar de texto a imagen.

Red Neuronal Convolucional (CNN)	Estas redes tienen la capacidad de operar con agrupaciones de datos con relaciones espaciales. Su arquitectura está formada por diferentes capas que componen el aprendizaje en un formato jerarquizado de características concretas para formar el modelado. En la visión y reconocimiento de elementos en imágenes, son las primeras capas de la red las encargadas de comprender los atributos básicos de los objetos —como esquinas y bordes—. Seguidamente, estos atributos se combinan de forma gradual con las capas posteriores para formar representaciones más complejas parecidas a los objetos de interés. Es entonces, en última instancia, que los atributos captados son empleados con el fin de predecir nuevos objetos de interés.
Codificador Automático	Anteriormente, se ha comentado el funcionamiento de los *embeddings* o incrustaciones. Los codificadores automáticos funcionan de una forma parecida a los *embeddings* de palabras dado que aportan representaciones de características densas de datos de entrada con la única diferencia de que no están limitados a datos de lenguaje natural; pueden aplicarse a cualquier entrada. Esta arquitectura se divide en dos etapas: una primera etapa de codificación, donde los datos que entran se constriñen en una baja dimensionalidad, y una segunda etapa contraria que decodifica. Los codificadores automáticos se suelen aplicar al aprendizaje no supervisado y la reducción de dimensionalidad.

Tabla 4: Principales arquitecturas de aprendizaje profundo (DL).

5. Desafíos para sistemas inteligentes basados en ML y DL

El escenario en el que nos encontramos contempla infinidad de posibilidades que combinan algoritmos arquitecturas, datos de entrenamiento y parámetros. A pesar de ello, uno de los grandes retos ante el que no encontramos es la usencia de pautas o pasos para elaborar modelos dedicados a problemas específicos que cumplan con cuatro aspectos esenciales: eficiencia, rendimiento, robustez y privacidad. En consecuencia, la construcción de modelos analíticos deviene un proceso fundamental para el éxito del sistema en cuestión. Suponiendo un modelo con una precisión de casi el cien por cien, pero que, sin embargo, el tiempo de decisión de clasificación es elevado, redunda en inutilidad puesto que en aplicaciones en el que el tiempo es una variable crucial se puede considerar un modelo con una precisión del 0%.

Es necesario destacar la diferencia de precisión entre resultados teóricos y resultados prácticos; las aplicaciones con un gran éxito de precisión en entornos controlados como laboratorios, no tiene por qué redundar en una precisión

práctica, puesto que pueden surgir nuevos factores no considerados en el entorno teórico. Por ende, los desarrolladores e ingenieros deben considerar la tesitura que introduce diferentes opciones arquitectónicas que se adaptan a distintos escenarios.

Capítulo IV. Simulación e implementación IDS

Con el siguiente estudio se pretende estudiar simulaciones de ataque en dispositivos con sistema operativo Contiki-NG y desarrollar un Sistema de Detección de Intrusos (IDS) para contribuir al campo de seguridad de redes IoT. Este trabajo se compondrá en cuatro fases: una primera fase de introducción a Cooja en donde se elaborará una pequeña simulación de ejemplo sobre ataques de inundación UDP, una segunda fase en la que configuraremos un escenario de ataque DIS-*flood* en una red IoT basada en RPL, una tercera fase de análisis de nuestras simulaciones y una cuarta fase de diseño de IDS.

1. Metodología y herramientas

La investigación se basará en un enfoque deductivo mediante un proceso de recopilación y análisis de datos. Se utilizará para tal propósito la máquina virtual VMware Workstation, el sistema operativo Contiki-NG y el simulador de redes Cooja.

VMware Workstation

Se trata de un software de virtualización capaz de crear una capa sobre el *hardware* del sistema anfitrión dividiendo el sistema en varios sistemas virtuales. Una máquina virtual o VM se ejecuta en una parte dedicada del sistema introduciéndose como un ordenador independiente con su propio sistema operativo. En otras palabras, una VM se puede entender como una representación basada en *software* de un sistema físico. Para nuestro experimento utilizaremos la máquina VMware Workstation 17 player. Disponible en [Recursos 1].

Contiki-NG

Es un sistema operativo implementado en código abierto para equipos del Internet de las Cosas. Su pila de protocolos es sencilla y dedicada a IoT —UDP, RPL, 6LoWPAN, IPv6, etc—. Además, como sistema integrado, utiliza UDP para más seguridad en el proceso comunicativo; utiliza Datagram Transport Layer Security (DTLS) en vez de Transport Layer Security (TLS). Disponible en [Recursos 2].

Cooja

Se presenta como un simulador de red IoT insertado en la distribución del SO Contiki-NG que nos permite simular escenarios sin tener que implementar *hardware* real para elaborar testeos. Nos facilita la emulación de motas organizadas en una red de sensores (WSN) de carácter inalámbrico. El motivo por el que utilizamos este programa es que se centra en dispositivos IoT de bajo consumo, cuya virtud es comúnmente buscada en tales equipos en redes modernas. Disponible en [Recursos 3].

2. Simulación DoS - UDP flood

En esta fase vamos a simular primeramente un Ataque de Denegación de Servicio (DoS) en donde un atacante enviará un número elevado de solicitudes a un servidor inundando el tráfico de la red impidiendo o dificultando el uso de los recursos de los otros hosts. Para tal propósito, implementaremos 8 nodos en topología de estrella que se dividirán en cliente, servidor y atacante. Los nodos 1-6 se corresponderán con los clientes, el nodo 7 será el servidor y el nodo 8 se comportará como un nodo malicioso. Tanto los clientes como el nodo maligno mandarán solicitudes UDP al servidor y éste responderá. Como resultado, obtendremos una inundación de peticiones UDP (*UDP flood*).

Para ello, primero codificaremos en el entorno de programación CodeLite [Recursos 18] tres archivos en lenguaje C: *cliente, servidor* y *atacante*. Estos archivos serán compilados y ejecutarán las correspondientes funciones de cada mota.

Observemos a continuación el código de cada uno de ellos:

2.1 Cliente

Véase en la carpeta de entraga: *src > simulación_1 > cliente.*

```c
#include "contiki.h"
#include "contiki-net.h"
#include <stdio.h>

#define SERVER_ADDR "fd00::1" // Dirección IPv6 del servidor
#define SERVER_PORT 1234      // Puerto del servidor

// Declaración del proceso cliente
PROCESS(client_process, "Client Process");

// Autoinicio del proceso cliente al arrancar el sistema
AUTOSTART_PROCESSES(&client_process);

static struct uip_udp_conn *client_conn; // Estructura para la conexión UDP del cliente
static char message[] = "Hola, Servidor!"; // Mensaje que se enviará al servidor

// Implementación del hilo de proceso del cliente
PROCESS_THREAD(client_process, ev, data)
{
    static struct etimer et; // Temporizador de eventos

    PROCESS_BEGIN(); // Inicio del proceso

    // Creamos una nueva conexión UDP sin dirección remota y puerto local asignado automáticamente
    client_conn = udp_new(NULL, UIP_HTONS(0), NULL);
    // Vinculamos la conexión UDP al puerto local asignado
    udp_bind(client_conn, UIP_HTONS(0));

    // Configuramos el temporizador para que expire cada 1 segundo
    etimer_set(&et, CLOCK_SECOND * 1);

    while (1)
    {
        // Esperamos hasta que el temporizador expire
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        // Reiniciamos el temporizador para el próximo evento
        etimer_reset(&et);

        // Enviamos el mensaje al servidor especificado en SERVER_ADDR y SERVER_PORT
        uip_udp_packet_sendto(client_conn, message, sizeof(message), SERVER_ADDR, UIP_HTONS(SERVER_PORT));

        printf("Paquete enviado al servidor\n");
    }

    PROCESS_END();
}
```

Figura 17: Código cliente.c

Por lo que respecta al cliente, se comportará como un nodo normal que mandará periódicamente mensajes UDP. Esto se desarrollará mediante un proceso enmarcado entre `PROCESS_BEGIN` y `PROCESS_END`. Utiliza el protocolo de red `RIME` —pila de red inalámbrica de SO Contiki— y entre cada envío esperará un intervalo de tiempo determinado por la línea `PROCESS_WAIT_EVENT_UNTIL` que espera hasta que expire el temporizador. Cada segundo el nodo no corrompido enviará periódicamente un mensaje al servidor.

2.2 Servidor

Véase en la carpeta de entraga: *src > simulación_1 > servidor.*

```c
#include "contiki.h"
#include "contiki-net.h"
#include <stdio.h>

#define SERVER_PORT 1234   // Puerto en el que el servidor escucha

// Declaración del proceso servidor
PROCESS(server_process, "Server Process");

// Autoinicio del proceso servidor al arrancar el sistema
AUTOSTART_PROCESSES(&server_process);

static struct uip_udp_conn *server_conn; // Se estructura para la conexión UDP del servidor

// Se implementa el hilo de proceso del servidor
PROCESS_THREAD(server_process, ev, data)
{
    PROCESS_BEGIN(); // Inicio del proceso

    // Se crea una nueva conexión UDP para escuchar en SERVER_PORT
    server_conn = udp_new(NULL, UIP_HTONS(SERVER_PORT), NULL);
    // Se vincula la conexión UDP al puerto especificado
    udp_bind(server_conn, UIP_HTONS(SERVER_PORT));

    while (1)
    {
        // Se espera hasta que ocurra un evento de TCP/IP (es decir, recepción de un paquete)
        PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event);
        // Se comprueba si se ha recibido un nuevo dato
        if (uip_newdata())
        {
            // Se imprime el mensaje recibido del cliente en la consola
            printf("Paquete recibido del cliente: %s\n", (char *)uip_appdata);
        }
    }

    PROCESS_END();
}
```

Figura 18: Código servidor.c

El servidor prestará atención al puerto asignado y esperará recibir mensajes UDP de los clientes para luego mandar acuses de recibo. El proceso del servidor se define en la línea `server_process` y se le indica al SO que dicho proceso inicie automáticamente. La creación de la conexión se codifica en la línea `server_conn` en el puerto especificado. Al igual que el cliente, el servidor opera con temporizadores para la escucha: se configura un temporizador que generará eventos cada diez segundos. Si recibe un paquete extraerá el mensaje del *payload* —o datos transmitidos útiles— del mismo. En cuando reciba un paquete mostrará un mensaje por pantalla.

2.3 Atacante

Véase en la carpeta de entraga: *src > simulación_1 > atacante.*

```c
#include "contiki.h"       // Incluye el núcleo de Contiki
#include "contiki-net.h"   // Incluye las funciones de red de Contiki
#include <stdio.h>

#define SERVER_ADDR "fd00::1" // Dirección IPv6 del servidor
#define SERVER_PORT 1234      // Puerto del servidor

// Declaración del proceso malicioso
PROCESS(malicious_process, "Malicious Process");

// Autoinicio del proceso malicioso al arrancar el sistema
AUTOSTART_PROCESSES(&malicious_process);

static struct uip_udp_conn *malicious_conn; // Estructura para la conexión UDP del proceso malicioso
static char message[] = "Ataque";          // Mensaje malicioso que se enviará al servidor

// Implementación del hilo de proceso del proceso malicioso
PROCESS_THREAD(malicious_process, ev, data)
{
    static struct etimer et; // Temporizador de eventos

    PROCESS_BEGIN(); // Inicio del proceso

    // Creamos una nueva conexión UDP sin dirección remota y puerto local asignado automáticamente
    malicious_conn = udp_new(NULL, UIP_HTONS(0), NULL);
    // Vinculamos la conexión UDP al puerto local asignado
    udp_bind(malicious_conn, UIP_HTONS(0));

    //Configuraramos el temporizador para que expire cada 0.1 segundos
    etimer_set(&et, CLOCK_SECOND / 10);

    while (1)
    {
        //Se espera hasta que el temporizador expire
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        // Reiniciamos el temporizador para el próximo evento
        etimer_reset(&et);

        // Se envía el mensaje malicioso al servidor especificado en SERVER_ADDR y SERVER_PORT
        uip_udp_packet_sendto(malicious_conn, message, sizeof(message), SERVER_ADDR, UIP_HTONS(SERVER_PORT));
        // Imprimimos un mensaje en la consola indicando que el paquete malicioso fue enviado
        printf("Paquete malicioso enviado al servidor\n");
    }

    PROCESS_END();
}
```

Figura 19: Código atacante.c

El atacante generará la ofensiva de denegación de servicio. Su proceso se compone en la línea `malicious_process` y se iniciará automáticamente. A diferencia de los clientes, el nodo malicioso mandará un paquete UDP corrupto cada décima de segundo, resultando en el colapso de la red.

Una vez cargados los códigos, estructuraremos la arquitectura de la red. Añadiremos 6 motas del tipo *Sky Mote* en disposición de elipse —aunque nos valdría cualquier otra con un radio no muy grande—. El motivo por el que elegimos este tipo de mota es debido a que son ideales para configuraciones iniciales dentro de una simulación Cooja.

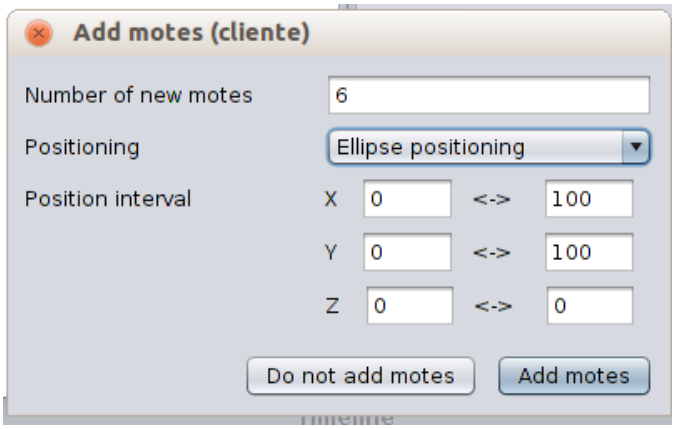


Figura 20: Creación motas

De manera análoga, crearemos el servidor y el nodo malicioso.

2.4 Escenario

En la Figura 21 se presenta la red de sensores IoT en la que se desarrollará el ataque DoS. Los nodos 1-6 verdes son los clientes, el nodo 7 amarillo central se corresponde con el servidor, mientras que el nodo 8 morado se corresponde con el nodo malicioso que realizará la ofensiva.

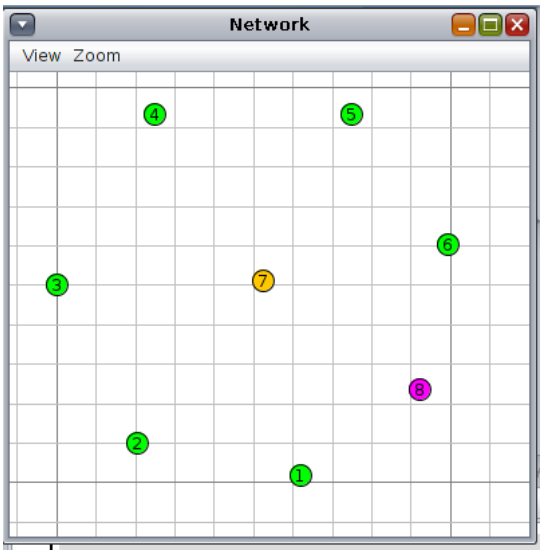


Figura 21: Escenario ataque DoS

El ataque dura un total de un minuto. En los primeros segundos, se ejecutan los procesos de cada mota:

```
00:01.164  ID:1   CSMA ContikiMAC, channel check rate 8 Hz, radio channel 65491
00:01.166  ID:1   Starting 'Client Process'
```

Figura 22: Inicio proceso nodo atacante

```
00:00.999  ID:7   CSMA ContikiMAC, channel check rate 8 Hz, radio channel 65491
00:01.001  ID:7   Starting 'Server Process'
```

Figura 23: Inicio proceso nodo servidor

```
00:00.753  ID:8   CSMA ContikiMAC, channel check rate 8 Hz, radio channel 65491
00:00.753  ID:4   CSMA ContikiMAC, channel check rate 8 Hz, radio channel 65491
00:00.755  ID:8   Starting 'Malicious Process'
```

Figura 24: Inicio proceso nodo atacante

Una vez iniciados los procesos, se empezarán a enviar los paquetes correspondientes:

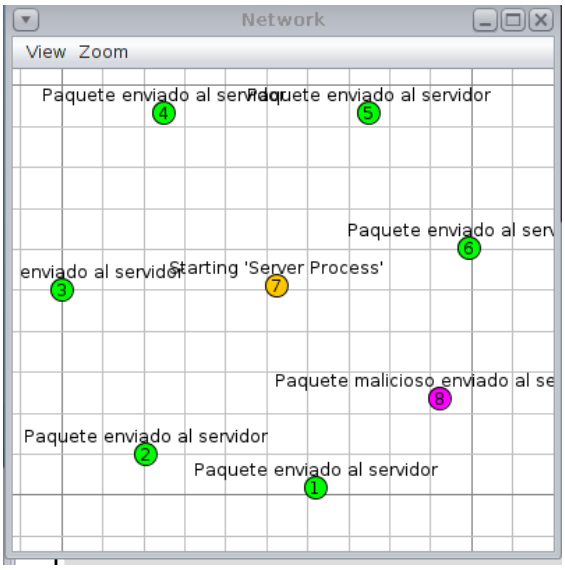


Figura 25: Operaciones de proceso

2.5 Análisis resultados UDP flood

Como se puede apreciar en la ventana de *output*, la salida se distribuye en 3 aspectos: tiempo, ID de la mota y mensaje.

La red se inunda de mensajes maliciosos al servidor. Concretamente, y como hemos mencionado anteriormente, se envía un paquete corrupto cada décima de segundo, es decir, cada 0.1 segundos, mientras que los paquetes de los clientes se envían cada segundo. Apreciemos el intervalo de tiempo 01.632 – 02.666 de

la Figura 26. En el tiempo 01.632 el cliente 4 envía un paquete normal al servidor; seguidamente, en el tiempo 01.666 el cliente 1 envía otro mansaje de la misma categoría; desde este punto, y hasta el tiempo 02.607, el nodo corrupto luchará para abordar los recursos inundando la red. No es hasta los tiempos 02.632 y 02.666 que las motas 4 y 1 volverán a poder utilizar los recursos.

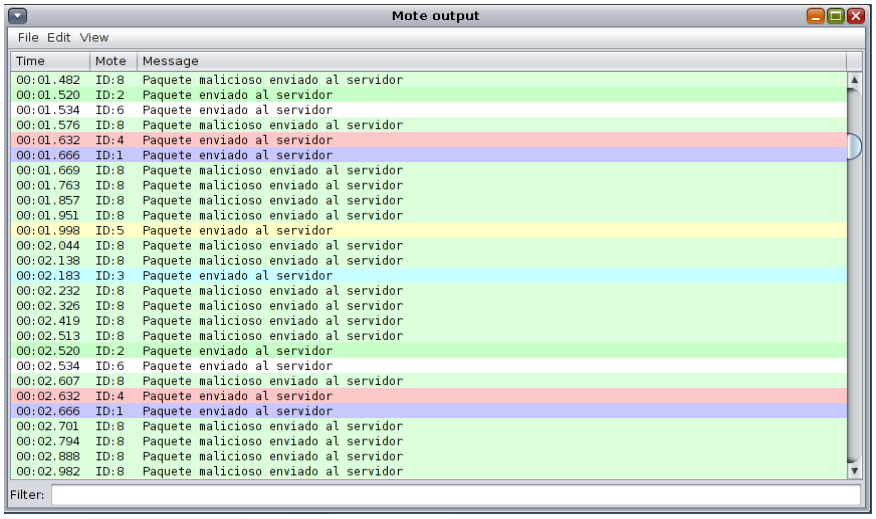


Figura 26: Mensajes enviados

3. Simulación DoS – DIS flood

Cabe destacar, sin embargo, que los puntos anteriores ilustran una simulación sencilla para la investigación. Tanto la disposición de la arquitectura de la red, número de motas, tipo de red, o tipo de ataque podría variar según otras intenciones de estudio.

A lo largo del grado hemos tenido la oportunidad de cursar asignaturas como Telecomunicaciones en el Sector del Transporte o Seguridad en Redes de Computadores, en las que pudimos analizar la posibilidad de que los ataques DoS pueden ocurrir en RPL (Routing Protocol for Low-Power and Lossy Networks). RPL es un protocolo de enrutamiento utilizado en redes de baja potencia y pérdida de paquetes estableciéndose como el preferido en redes IoT. En la capa de red, el protocolo de enrutamiento RPL se fundamenta como el más conocido para 6LoWPAN. Destaca por su adaptación a la red y proporcionar rutas alternativas, pues está formulado para construir topologías de red. RPL implementa distintos tipos de mensajes de control como *DODAG Information Solicitation* (DIS), DODAG *Information Object* (DIO), *Destination Adversiment Object* (DAO) y *Destination Advertisement Object Acknoledgment* (DAO-ACK). Las siglas DODAG significa Grafo Acíclico Dirigido Orientado al Destino.

A colación de lo anterior, corremos una nueva simulación DoS, pero con la peculiaridad de que dicho escenario está configurado en la pila RPL basada en 6LoWPAN. En este trabajo, sin embargo, no se indagará mucho en la función de

la red IoT, pues nuestro estudio es aplicable y compatible para muchos ámbitos como: *smart cities*, hogar, gestión de tráfico, asistencia en emergencias, etcétera. No obstante, para obtener una mejor comprensión del lector, consideraremos de ahora en adelante que nuestra red está dedicada, por ejemplo, a un sistema de riego inteligente. Así pues, en la Tabla 5 se muestran las configuraciones de la red.

Capa de Transporte	La misma: UDP
Capa de Adaptación	6LoWPAN
Capa de Red	IPv6-ICMP-RPL
Protocolo de enrutamiento	RPL
Nodo	Z1 dado que soporta IEEE802.15.4 que es ideal para sistemas de baja potencia.
Protocolo MAC	CSMA
Área	200m^2
Número de nodos	12 en total, entre ellos: 11 clientes, un nodo *sink* y un atacante
Tiempo de simulación	60 minutos

Tabla 5: Características de la simulación.

A diferencia de lo anterior, correremos una versión de red formada por un nodo *sink* y diez clientes; también añadiremos un nodo malicioso que realizará el ataque de denegación de servicio DIS *flood*. Dicho nodo enviará un número muy grande de paquetes DIS a los nodos víctima con el objetivo de entorpecer el tráfico de la red, anularlo o directamente agotar la batería de los nodos. El nodo *sink* se corresponderá con la mota principal y recibirá información sobre otros nodos de la red a la vez que actuará como servidor UDP donde guardará información de los clientes proveniente del sensor, como temperatura y humedad. También deberá reenviar los datos al *router* 6LoWPAN donde se extraerán las informaciones relevantes, esto que veremos más adelante. El nodo cliente, por su parte, y como hemos introducido anteriormente, tendrá el objetivo concreto de obtener datos de humedad y temperatura, así como otros datos de valor que veremos más adelante.

Todos los sensores se implementarán mediante la mota Z1 estableciendo así una red homogénea. Los sensores en cuestión serán los SHT21 y SHT25. Por otra parte, consideraremos que la conexión entre el *router* e internet se llevará a cabo mediante un enlace Ethernet. Veamos a continuación el resumen de características de todos los nodos:

Mota	Z1
RAM	8kb
Estándar	IEEE 802.15.4
CPU	16-bit RISC CPU @16MHz
Sensor	SHT21, SHT25

Tabla 6: Características de nodo.

En este punto, empezamos creando las motas. Para ello, utilizaremos los archivos públicos que ofrece Cooja y tenemos a nuestra disposición. Estos son:

3.1 Nodo Sink

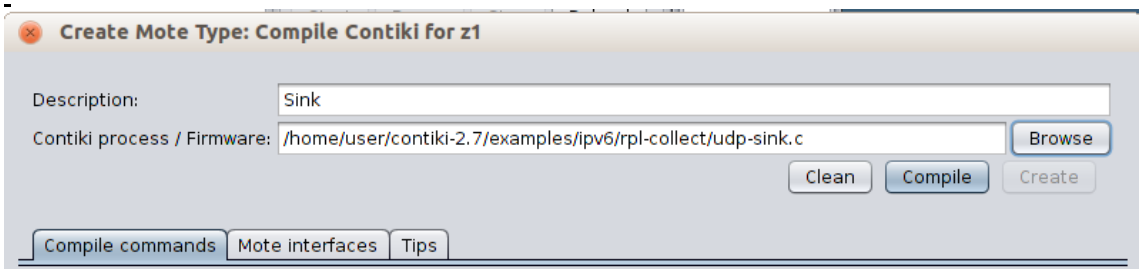


Figura 27: Creación Nodo Sink simulación DIS flood

A continuación, comentamos las partes más importantes del archivo `udp-sink` [Recursos 4], con el que implementamos la mota Sink:

```
/*creamos estructura para la conexión*/
static struct uip_udp_conn *server_conn;

PROCESS(udp_server_process, "UDP server process");
AUTOSTART_PROCESSES(&udp_server_process,&collect_common_process);
/*---------------------------------------------------------------------------*/
/*manejamos los eventos de la pila TCP/IP, si hay nuevos disponibles, llamamos a la
función colect*/
static void tcpip_handler(void)
{
  uint8_t *appdata;
  rimeaddr_t sender;
  uint8_t seqno;
  uint8_t hops;

  if(uip_newdata()) {
    appdata = (uint8_t *)uip_appdata;
    sender.u8[0] = UIP_IP_BUF->srcipaddr.u8[15];
    sender.u8[1] = UIP_IP_BUF->srcipaddr.u8[14];
    seqno = *appdata;
    hops = uip_ds6_if.cur_hop_limit - UIP_IP_BUF->ttl + 1;
    collect_common_recv(&sender, seqno, hops,
                        appdata + 2, uip_datalen() - 2);
  }
}
/*---------------------------------------------------------------------------*/
/*imprimimos las direcciones IPv& del servidor*/
static void print local addresses(void)
{
  int i;
```

```
  uint8_t state;

  PRINTF("Server IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(state == ADDR_TENTATIVE || state == ADDR_PREFERRED) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if (state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
/*creamos el hilo principal donde inicializamos el servidor, creamos una conexión y
entramos en un bucle infinito donde manejamos la pila TCP/IP*/
PROCESS_THREAD(udp_server_process, ev, data)
{
  uip_ipaddr_t ipaddr;
  struct uip_ds6_addr *root_if;

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  SENSORS_ACTIVATE(button_sensor);

  PRINTF("UDP server started\n");

#if UIP_CONF_ROUTER
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
  /* uip ds6 set addr iid(&ipaddr, &uip lladdr); */
  uip_ds6_addr_add(&ipaddr, 0, ADDR_MANUAL);
  root_if = uip_ds6_addr_lookup(&ipaddr);
  if(root_if != NULL) {
    rpl_dag_t *dag;
    dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(uip_ip6addr_t *)&ipaddr);
    uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
    rpl_set_prefix(dag, &ipaddr, 64);
    PRINTF("created a new RPL dag\n");
  } else {
    PRINTF("failed to create a new RPL DAG\n");
  }
#endif /* UIP_CONF_ROUTER */

  print_local_addresses();

/* El sink de datos se ejecuta con un ciclo de trabajo del 100% para garantizar altas
tasas de recepción de paquetes*/


  NETSTACK_RDC.off(1);

  server_conn = udp_new(NULL, UIP_HTONS(UDP_CLIENT_PORT), NULL);
  udp_bind(server_conn, UIP_HTONS(UDP_SERVER_PORT));

  PRINTF("Created a server connection with remote address ");
  PRINT6ADDR(&server_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n", UIP_HTONS(server_conn->lport),
         UIP_HTONS(server_conn->rport));

  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    } else if (ev == sensors_event && data == &button_sensor) {
      PRINTF("Initiaing global repair\n");
      rpl_repair_root(RPL_DEFAULT_INSTANCE);
    }
  }

  PROCESS_END();
}
```

3.2 Cliente

Seguidamente, creamos 10 nodos cliente:

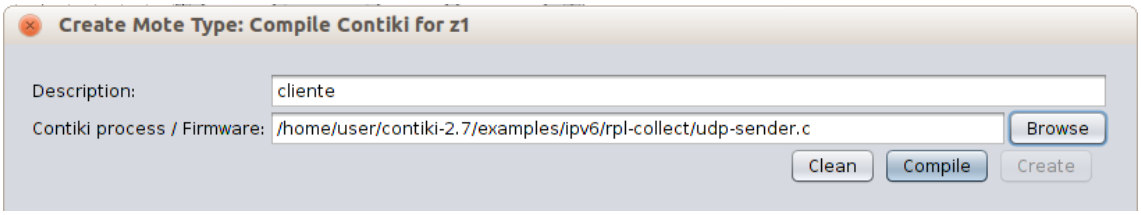


Figura 28: Creación Nodo Cliente simulación DIS flood

Seguidamente, comentamos las partes más importantes del archivo `udp-sender` [Recursos 5], con el que implementamos la mota cliente:

```
/*declaramos estructura conexión cliente*/
static struct uip_udp_conn *client_conn;
static uip ipaddr t server ipaddr;

/*---------------------------------------------------------------------------*/
/*enviamos mensaje al servidor que contendrá información sobre el padre, la métrica de la
ruta y el número de vecinos*/
void collect_common_send(void)
{
  static uint8_t seqno;
  struct {
    uint8_t seqno;
    uint8_t for_alignment;
    struct collect_view_data_msg msg;
  } msg;
  /* struct collect_neighbor *n; */
  uint16_t parent_etx;
  uint16_t rtmetric;
  uint16_t num_neighbors;
  uint16_t beacon_interval;
  rpl_parent_t *preferred_parent;
  rimeaddr_t parent;
  rpl_dag_t *dag;

  if(client_conn == NULL) {
    /* Not setup yet */
    return;
  }
  memset(&msg, 0, sizeof(msg));
  seqno++;
  if(seqno == 0) {
    /* Wrap to 128 to identify restarts */
    seqno = 128;
  }
  msg.seqno = seqno;

  rimeaddr_copy(&parent, &rimeaddr_null);
  parent_etx = 0;

  /* suponemos que solo tenemos una instancia*/
  dag = rpl_get_any_dag();
  if(dag != NULL) {
    preferred_parent = dag->preferred_parent;
    if(preferred_parent != NULL) {
      uip_ds6_nbr_t *nbr;
      nbr = uip_ds6_nbr_lookup(rpl_get_parent_ipaddr(preferred_parent));
      if(nbr != NULL) {
       /*utilizamos partes de la adress Ipv6 como la dirección padre en orden de
bytes                          invertido*/
        parent.u8[RIMEADDR_SIZE - 1] = nbr->ipaddr.u8[sizeof(uip_ipaddr_t) - 2];
        parent.u8[RIMEADDR_SIZE - 2] = nbr->ipaddr.u8[sizeof(uip_ipaddr_t) - 1];
        parent_etx = rpl_get_parent_rank((rimeaddr_t *) uip_ds6_nbr_get_ll(nbr)) / 2;
```

```
      }
    }
    rtmetric = dag->rank;
    beacon_interval = (uint16_t) ((2L << dag->instance->dio_intcurrent) / 1000);
    num_neighbors = RPL_PARENT_COUNT(dag);
  } else {
    rtmetric = 0;
    beacon_interval = 0;
    num_neighbors = 0;
  }

  /* num_neighbors = collect_neighbor_list_num(&tc.neighbor_list); */
  collect_view_construct_message(&msg.msg, &parent,
                                 parent_etx, rtmetric,
                                 num_neighbors, beacon_interval);

  uip_udp_packet_sendto(client_conn, &msg, sizeof(msg),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
/*---------------------------------------------------------------------------*/
/*establecemos la dirección global del cliente y la dirección del servidor*/
static void set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);

  /* set server address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);

}
/*---------------------------------------------------------------------------*/
/*inicializamos el cliente, creamos una conexión UDP y manejamos los datos de la pila*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  PROCESS_BEGIN();

  PROCESS_PAUSE();

  set_global_address();

  PRINTF("UDP client process started\n");

  print_local_addresses();

  /*nueva conexión con el host remoto*/
  client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
  udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
        UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    }
  }
  PROCESS_END();
}
```

3.3 Malicioso

Finalmente, creamos el nodo malicioso:

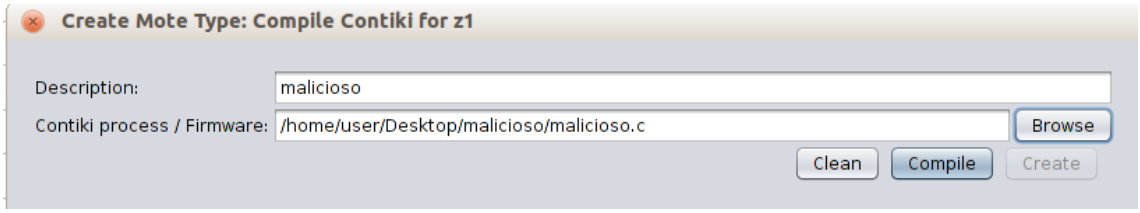


Figura 29: Creación Nodo Malicioso simulación DIS flood

Por lo que respecta al código del atacante, es parecido al `udp-sender`, pero con algunas modificaciones. A continuación, comentamos las partes modificadas del archivo `malicioso` con el que implementamos la mota maligna.

Véase el códgo entero en la carpeta de entraga: *src > simulación_2 > malicicoso_RPL.*

```
//Mandaremos un paquete DIS cada segundo, es decir, cada 1000 milisegundos
#define RPL_DIS_INTERVAL 1000

//Mandaremos mensajes DIS inmediatamente al iniciarse
#define RPL_DIS_START_DELAY 0
```

Como se aprecia en el código superior, definiremos una constatne para enviar mensajes DIS cada segundo, que es lo que buscamos: inundar la red de dichos mensajes. También estableceremos un delay a 0 para que dichos paquetes se envíen inmediatamente.

```
// Enviamos paquetes DIS en el intervalo especificado
   if (clock_time_exceeded(timer_from_now(RPL_DIS_INTERVAL))) {
     // Constructor mensaje DIS
     struct rpl_dis_hdr dis;
     dis.type = RPL_DIS_TYPE_REQUEST;
     dis.len = 0;

     //Mandamos DIS
     uip_udp_packet_sendto(client_conn, &dis, sizeof(dis),
       &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));

     PRINTF("Sent DIS message\n");
   }
```

Por otra parte, mediante la sentencia `clock_time_exceeded` comprobamos si ha pasado el intervalo específico por `RPL_DIS_INTERVAL`. Si ha pasado el tiempo indicado, entraremos en la sentencia `if` en donde se creará el constructor del paquete que vayamos a enviar. Para terminar, mediante la operación `uip_udp_packet_sendto` se envía el mensaje DIS estableciendo la conexión, el tamaño del mensaje, la IP del servidor y su puerto.

Finalmente, obtenemos la siguiente topología de red en la que se elaborará el ataque de denegación de servicio *DIS flood*:

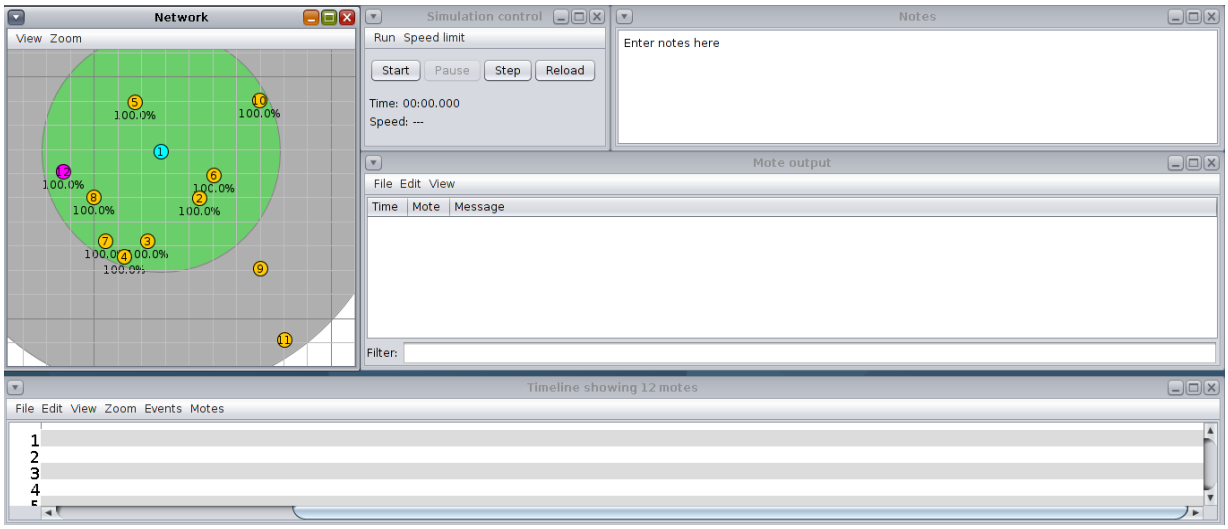


Figura 30: Topología de red simulación DIS flood

Para ver el rango de cobertura del nodo nodo *sink* (1) seleccionaremos la pesataña *view -> radio enviroment (UDGM).* En la Figura inferior vemos más de cerca nuestro caso de estudio.

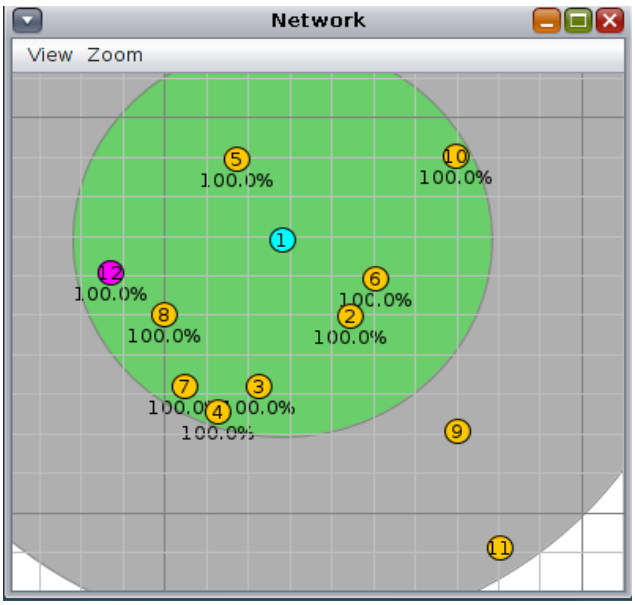


Figura 31: Distribución de nodos

Como se puede apreciar en la Figura 32, desplegamos un nodo *sink* — número 1, color azul; más tarde verde, pues lo tenemos seleccionado—, 10 nodos normales

— 2 a 11, color amarillo— y un nodo malicioso —número 12, color púrpura—. El rango del nodo malicioso abarcará los nodos 3, 4, 5, 7 y 8 puesto que están cubiertos por la zona verde.

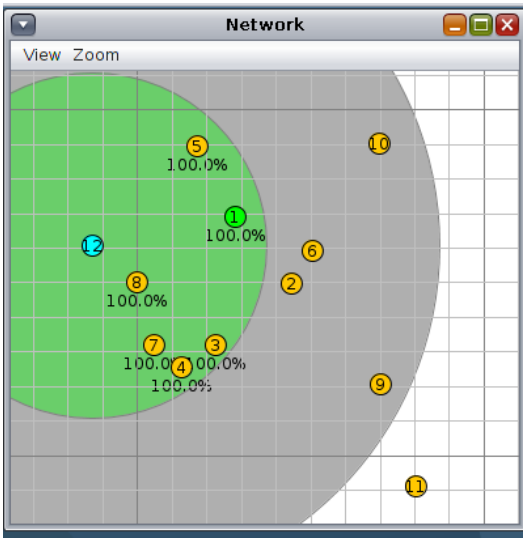


Figura 32: Nodo Sink (1), nodo malicioso (12) y nodos normales (2-11)

Una vez en este punto, corremos la simulación pulsando *start*. Lo primero que veremos en la ventana de *output* será el establecimiento de la red RPL por parte de la WSN (traducido al español, Red Inalámbrica de Sensores). En la Figura 33, vemos como los nodos van creando uno a uno la conexión con el nodo Sink.

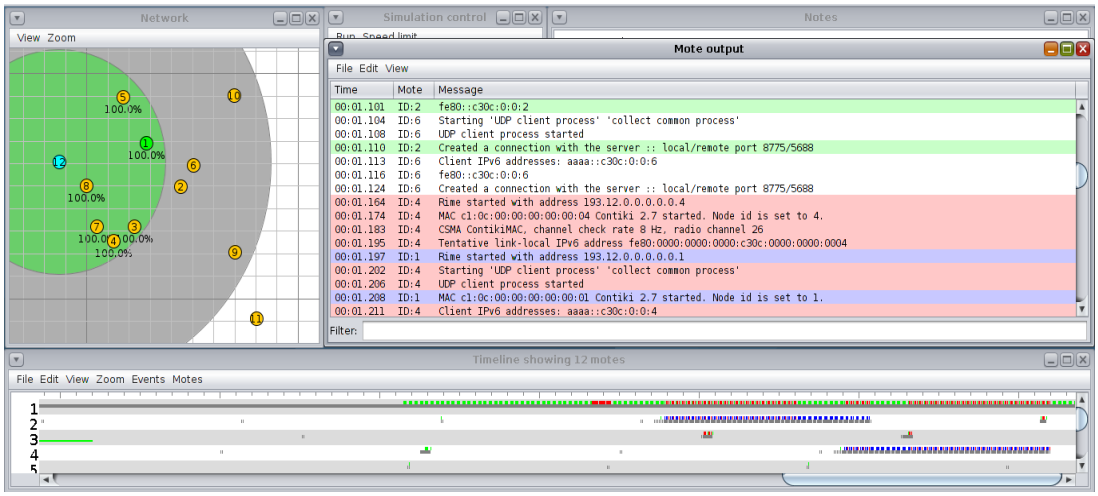


Figura 33: Creación de las conexiones con el nodo Sink

Por ejemplo, en la Figura inferior, si observamos el nodo 3, en el momento 1.715 se inicia el protocolo Rime con la dirección 193.12.0.0.0.0.3. En el momento 1.725 nos indica que el dispositivo con la dirección MAC c1:00:00:00:00:03 se ha iniciado mediante Contiki 2.7. Seguidamente, en 1.734 muestra los detalles de CSMA y la tasa de chequeo de canal. En 1.746 se asigna una dirección IPv6 tentativa de enlace local. Seguidamente, se inicia el proceso UDP; la dirección del cliente es aaaa::c30c:0:0:3. Finalmente, en el momento 1.773 se crea la conexión con el servidor en los puertos locales/remotos 8775 y 5688 respectivamente. Como la mota con ID 3 es la última en establecer conexión podemos decir que la red RPL tarda 1.773 segundos en formarse.

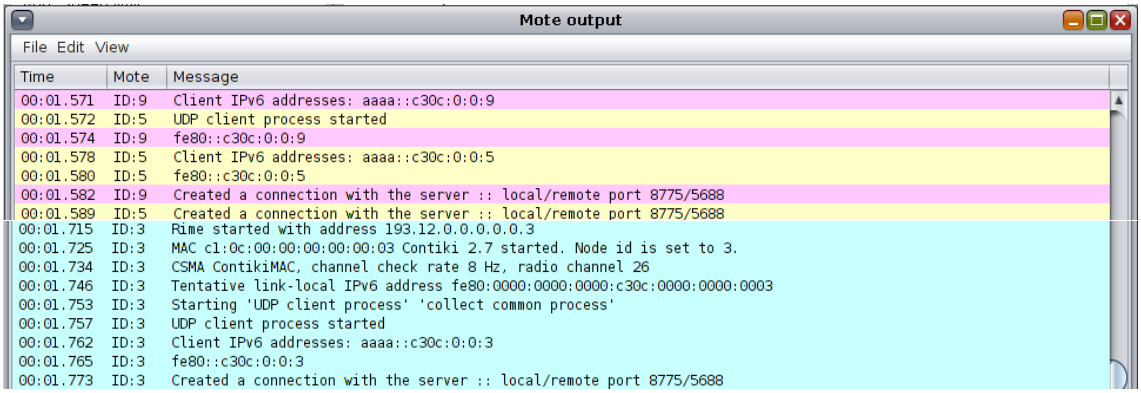


Figura 34: Tiempos de proceso y conexión

Una vez todos los nodos hayan establecido la conexión empezarán a enviar tráfico de datos al nodo *sink.* Para ver la información que transmiten los nodos usaremos el *Collect View* de Cooja que nos permitirá obtener distintas informaciones. En nuestro caso, nos centraremos en el gráfico *Power* dado que, como sabemos, los ataques de denegación de servicio utilizan muchos recursos de la red entorpeciendo el tráfico de la misma consumiendo mucha energía de los nodos. Por tanto, prevemos un gran aumento del consumo promedio de energía. Para recolectar los datos y poder evaluar las consecuencias de nuestro ataque nos dirigimos al nodo *sink*, hacemos clic derecho y seleccionamos *Mote tools for Z1 1-> Collect View*. Véase la pestaña *Collect View*:

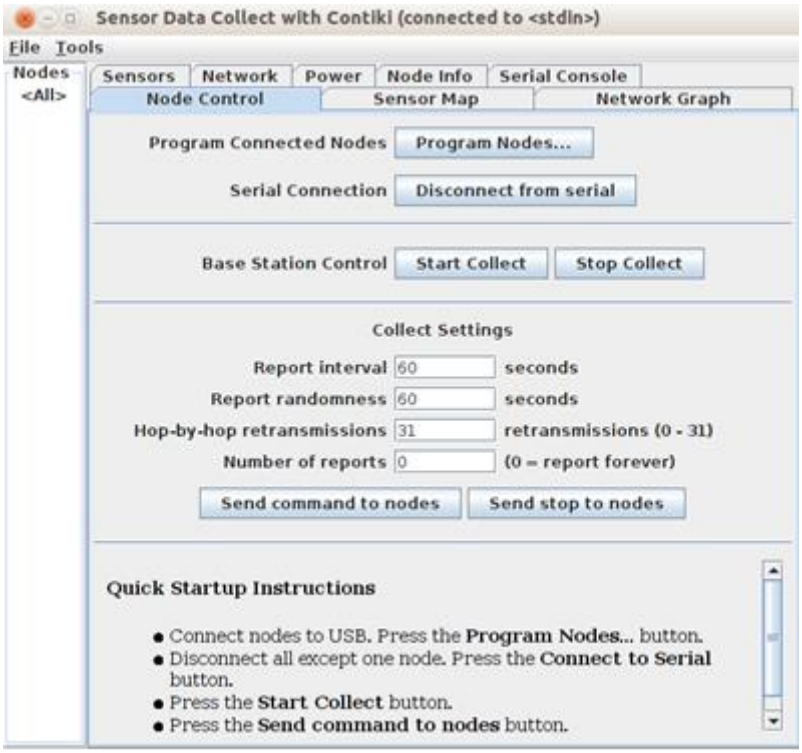


Figura 35: Configuración colección de datos

En este punto, seleccionamos *Start Collect* y *Send Command to Nodes*. A continuación, vemos como se empecerán a enviar datos:

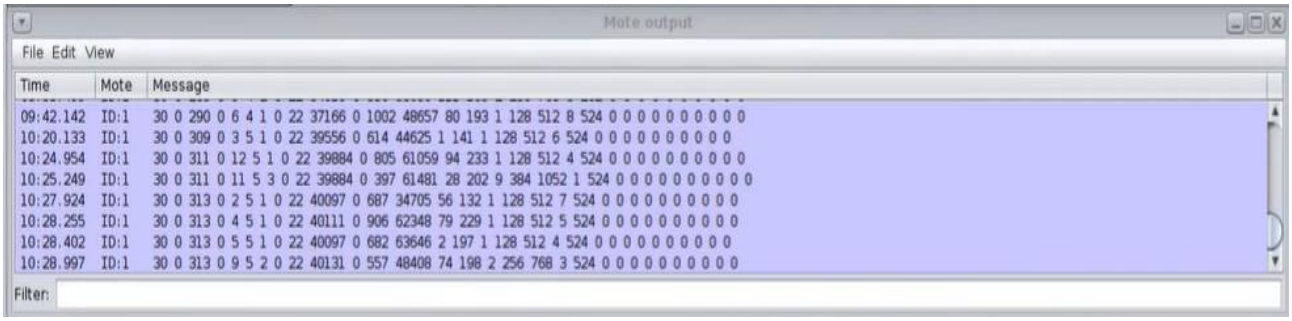


Figura 36: Envío de datos

Con el fin de conseguir la mejor precisión, desarrollaremos la simulación durante un total de 60 minutos obteniendo así los mejores resultados. Seguidamente, compararemos los gráficos de energía de la ejecución de nuestra red sin el nodo malicioso —sustituyéndolo por uno normal— y de la ejecución con el nodo malicioso integrado. Para eliminar el nodo y obtener los resultados sin ataque, simplemente haremos clic derecho sobre la mota -> *delete all motes of type*: *malicioso*, crearemos otro nodo cliente y procederemos como hemos indicado.

4. Análisis resultados DIS flood

Veamos, pues, las diferencias de energía antes del ataque y después del ataque:

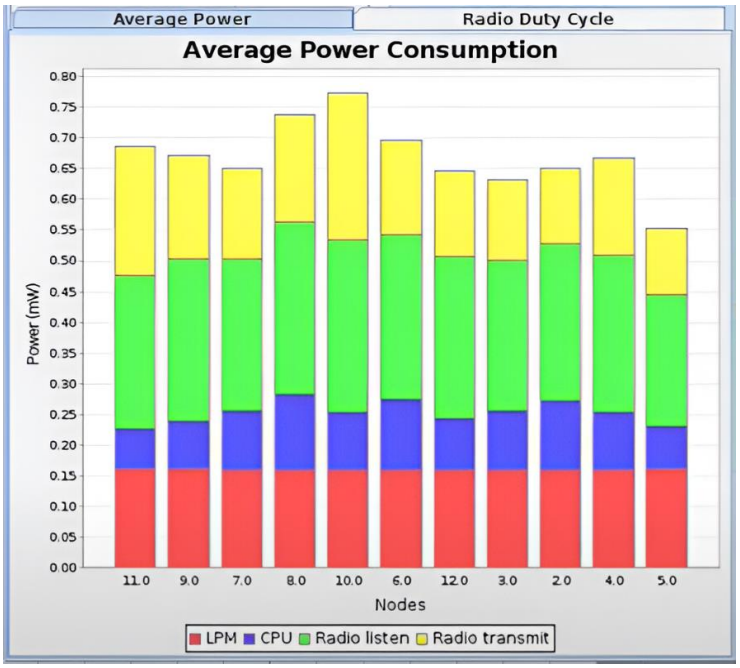


Figura 37: Consumo medio de energía de los nodos antes del ataque

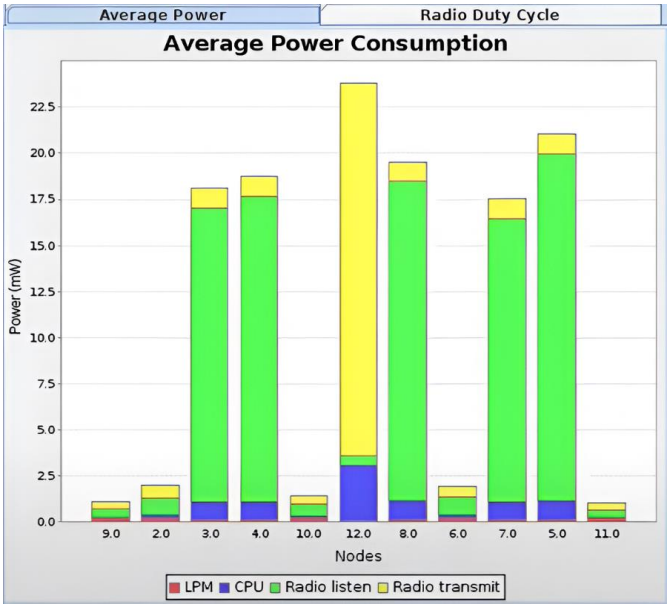


Figura 38: Consumo medio de energía de los nodos después del ataque

Estudiemos bien los cambios anteriores.

En ambas figuras se nos representa el consumo medio de energía de los nodos 2 – 12 siendo el último el nodo maligno. El eje x nos muestra las motas, mientras que la energía se muestra en el eje y en mW (miliwatt). Además, en ambos gráficos, el consumo se *divide LPM, CPU, Radio listen* y *Radio transmit*. Respecto a la Figura 37, observamos que los todos los nodos consumen entre un 0.55 mW y 0.77mW con un mismo consumo de LPM y parecidos consumos de CPU y Radio, lo cual nos indica un funcionamiento normal de la red. El mayor consumo de energía antes del ataque lo copa el nodo 10 con 0.75 mW. Siguiendo con el análisis, vemos en la Figura 38, después del ataque, un aumento considerable del consumo de energía. Destacamos seis nodos: el nodo 3, 4, 12, 8, 7 y 5. La mota 12 muestra un aumento considerable del consumo de energía de *Radio transmit* superando los 22.5 mW, lo cual tiene sentido pues el nodo atacante inserta una gran cantidad de mensajes DIS en nuestra red. En cuanto a los nodos 3, 4, 8, 7 y 5, vemos un gran aumento de la energía en *Radio listen*, cuya presencia es entendible porque son las motas más cercanas a la mota maligna, entonces, recibirán una gran cantidad de tráfico. El consumo de energía también aumenta en el resto de nodos, pero en un factor más bajo, el nodo 11, por ejemplo, será el que experimentará una menor subida al ser el nodo más lejano. Por todo lo anterior, y en consecuencia, concluimos que después del ataque se produce un gran aumento del consumo medio de energía de los nodos de la red, lo cual coincide con lo augurado en los apartados anteriores.

5. Implementación IDS

En la Figura 39 se presenta el esquema general de la arquitectura de detección. Como se puede apreciar, está dividido en 4 módulos: Módulo de Colección de Datos, Módulo de Clasificación, Módulo de Detección y Módulo IDS.

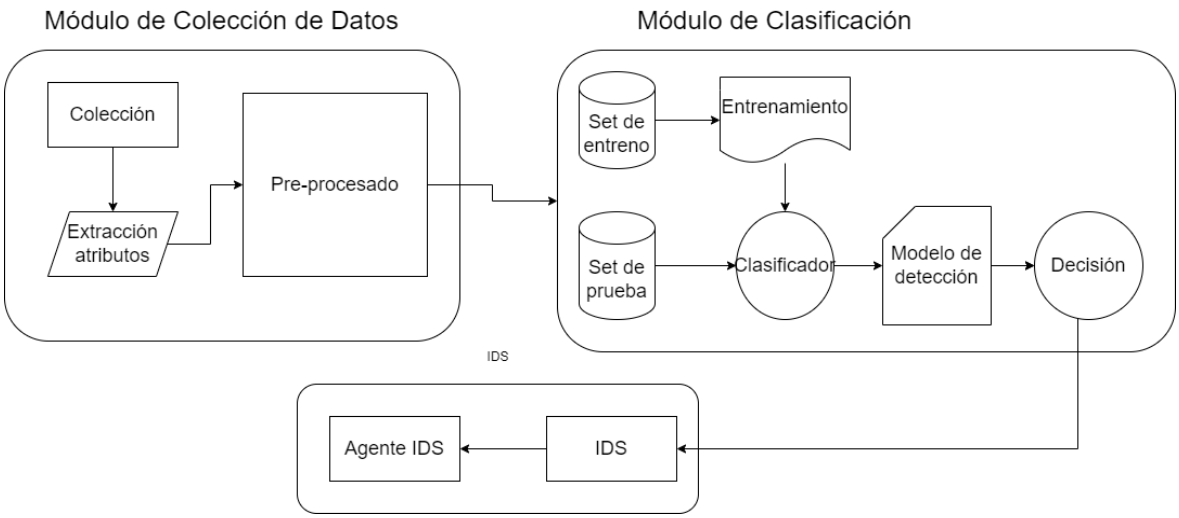


Figura 39: Esquema general IDS

A su vez, estableceremos dos fases: una fase de pre-aprendizaje y una fase de post-aprendizaje. En la primera fase, el Módulo de Colección de Datos (DCM) es utilizado para recopilar todos aquellos datos e informaciones *offline*, para entrenar los algoritmos que vamos a utilizar; una vez entrenado, se testeará y estudiará cuál modelo es mejor para implementar en nuestro IDS. Por otro lado, en la segunda fase, la fase de Post-Aprendizaje, el DCM se dedicará a inspeccionar, velar y monitorear el modelo de aprendizaje automático entrenado y seleccionado en la fase anterior.

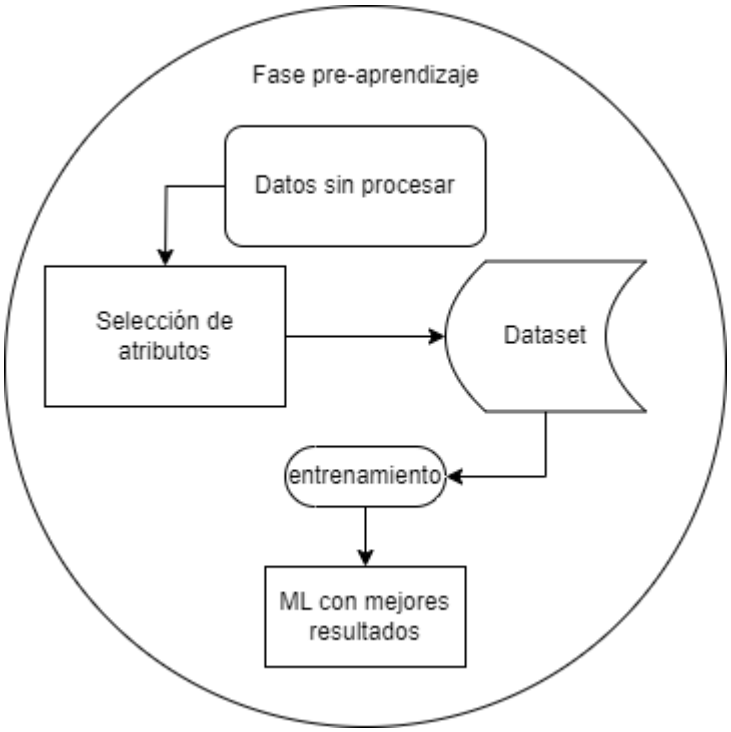


Figura 40: Fase pre-aprendizaje

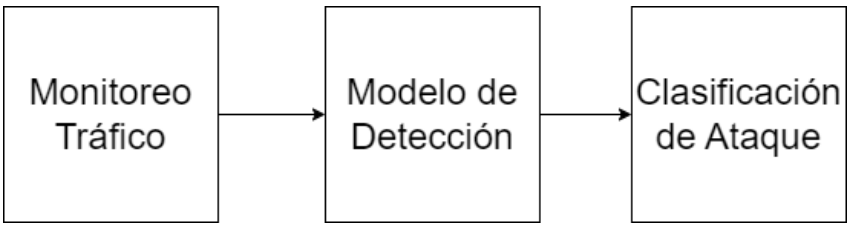


Figura 41: Fase post-aprendizaje

6. Módulo de Colección

Como se ha introducido anteriormente, la pila de informaciones generada por el Módulo de Copilación de Datos (DCM) se utilizará para entrenar y probar nuestro algoritmo de aprendizaje automático. En concreto, extraeremos los datos en función de 3 características: características de Capa Física, de Capa de Red y Capa de Aplicación. Estos datos, a su vez deberán ser purgados de aquellos innecesarios que dificulten el aprendizaje de nuestro modelo. El objetivo de este módulo es, pues, adquirir atributos propios de una red 6LoWPAN como: el número de secuencia del paquete, el tiempo de recepción y envío, dirección IP de origen, ID del nodo, dirección del nodo padre, *delay* y número de Mensajes DIS, entre otros que veremos a continuación. Una vez entrenado nuestro modelo con las informaciones obtenidas, analizaremos los resultados y estableceremos qué método es el más adecuado para implementar en nuestro IDS.

6.1 Características de la Capa Física

Se extraerán características que tengan una estrecha relación con la capa física como señales de la capa MAC.

6.2 Características de la Capa de Red

Las características extraídas de la Capa de Red serán muy útiles en nuestra detección, pues están muy relacionadas con los ataques DoS/DDoS. Extraeremos datos como el número de paquetes de Solicitud de Información (DIS) que son enviados al nodo para unirse a la red. En adición, extraeremos atributos relacionados con el protocolo RPL.

6.3 Características de la Capa de Aplicación

En esta categoría extraeremos atributos como el nivel de energía del nodo —que se verá aumentados por los ataques—, datos propios de la red, que, siguiendo nuestro ejemplo de aplicación para un sistema de riego inteligente, serán datos sobre la temperatura, humedad o energía del nodo —las informaciones de esta capa podrán variar en función de la función de la red—. La energía de cada nodo, por ejemplo, nos permitirá estudiar cómo afecta el ataque a nuestra red.

6.4 Selección de Características

Para extraer las características, proponemos el uso del sensor CC2420 [Recursos 11] que transmitirá los datos al Módulo de Colección. Una vez lo haya recibido, éste extraerá las características según su capa. En la Figura 42, podemos ver el proceso de extracción. Como vemos, una vez se hayan recopilado los datos deseados de cada capa, obtendremos una pila de datos —base de datos— que será retransmitida al siguiente módulo: el Módulo de Clasificación.

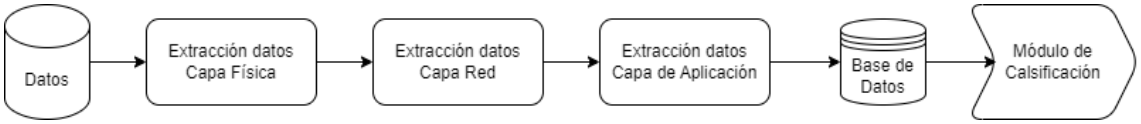


Figura 42: Proceso selección de características

Dada la naturaleza de nuestra red, una vez agrupados los atributos obtendremos la siguiente tabla:

Abreviatura	Descripción
No.	Número de secuencia del paquete
recTime	Tiempo de recepción del paquete
sendTime	Tiempo de envío del paquete
sendrIP	Dirección IP de origen
nodeID	ID del nodo
Delay	Tiempo recibido – tiempo enviado
numSentDIO	Conteo de mensajes DIO enviados
numSentDIS	Conteo de mensajes DIS enviados
residualEnergy	Energía residual del nodo en el momento del envío
label	Etiquetas para diferentes ataques
rank	Rango RPL
power	Nivel de potencia
temp	Temperatura
humidity	Humedad
rxdbm	Señal DBM recibida
txdbm	Señal DBM transmitida

Tabla 7: Características extraídas.

7. Módulo de Clasificación y Detección

En la Figura 43, se puede apreciar de cerca el Módulo de Clasificación y Detección que dividirá los datos recogidos en 2 sets: uno dedicado al entrenamiento de nuestros modelos y otro dedicado al testeo. Una vez realizadas estas operaciones, se validará el algoritmo que finalizará aportando una decisión dependiendo de nuestro caso: estado de ataque o estado normal.

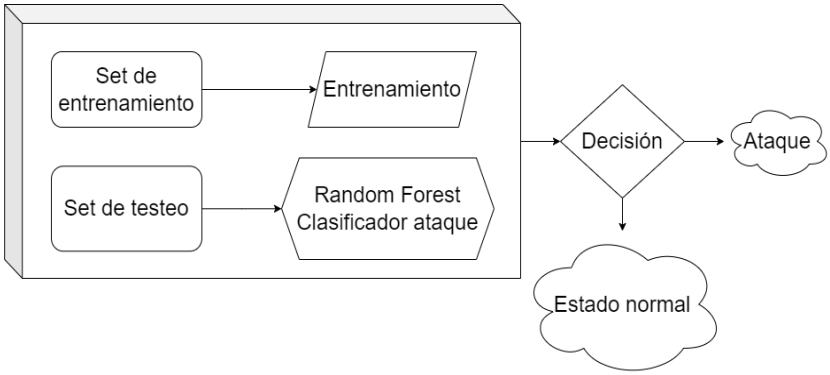


Figura 43: Módulo de Clasificación y Detección

A colación de lo anterior, y como hemos mencionado con anterioridad, estableceremos dos fases para entrenar nuestro modelo de aprendizaje: una primera Fase de Pre-aprendizaje y una segunda Fase de Post-aprendizaje.

7.1 Fase de Pre-aprendizaje

En la Fase de Pre-aprendizaje nuestro algoritmo se entrena a partir de los datos obtenidos. Una vez entrenado, se selecciona el modelo más adecuado para nuestro escenario. Veamos, pues, el proceso completo.

Primero debemos seleccionar aquellos modelos a entrenar. Si bien es cierto que en la actualidad encontramos una gran variedad de algoritmos disponibles que ofrecen soluciones a ofensivas, nosotros, para este estudio. elegiremos los dos de uso más frecuente en lo que se refiere a detección de intrusos: Support Vector Machine y Random Forest.

7.1.1 Support Vector Machine (SVM)

Es un método supervisado para la clasificación y regresión. Dado un conjunto de datos, cada uno se marca en una de dos categorías. Un algoritmo SVM construye un modelo que asigna nuevos ejemplos a una categoría u otra. Realiza la clasificación encontrando en el hiperplano que maximiza el margen entre las dos clases. Cada vector de soporte son los puntos de datos más cercanos al hiperplano, pueden ser puntos de datos más difíciles de clasificar, y dan mayor contribución a la determinación de la posición del hiperplano.

7.1.2 Arboles de Decisión (Random Forest)

Es un método de aprendizaje para la clasificación, regresión y otras tareas de operan mediante la construcción de una multitud de árboles de decisión en el momento del entrenamiento y generando la clase que es el modo de las clases o predicción media de los árboles individuales. Utilizan una

estructura de árbol para representar una serie de decisiones posibles y sus resultados. Es decir, se crea un conjunto de árboles a partir de un subconjunto aleatoriamente seleccionado de los datos de entrenamiento.

Los modelos anteriores se detallarán más adelante con mayor profundidad.

En este punto, entramos de lleno en la parte de aprendizaje, en donde los algoritmos aprovecharán todos los datos recopilados anteriormente descritos. Seguidamente, se produce la etapa de verificación para estudiar los resultados de los modelos. Finalmente, y basándonos en los resultados de la etapa anterior, se elige un solo modelo para implementar en nuestro IDS. Veamos, pues, la etapa de verificación:

7.1.3 Verificación de Algoritmo

Para testear el rendimiento del algoritmo de aprendizaje usaremos la matriz de confusión en donde compararemos los valores predichos con los valores reales. Las filas se corresponderán con los valores reales en función de clase y las columnas se corresponderán con los valores de predicción.
Los parámetros son:

- Verdadero Positivo (VP): muestras normales identificadas correctamente.

- Falso positivo (FP): muestras identificadas como ataque normal (no identificadas correctamente).

- Verdadero Negativo (VN): muestras identificadas correctamente como ataque.

- Falso Negativo (FN): muestras incorrectamente identificadas como normales siendo, en realidad, ataques reales.

De este modo, calcularemos las siguientes métricas:

Exactitud
Es una información útil para obtener la puntuación general del algoritmo: Se calcula como:

$$Exactitud = \frac{Número\ correcto\ de\ precisiones}{Número\ total\ de\ predicciones}$$

(1)

Precisión
Surge de la necesidad de métricas más precisas para construir un modelo de aprendizaje mejorado. Lo calculamos como:

$$Precisión = \frac{VP}{VP + FP}$$

(2)

Recall

Es la proporción entre el número de ataques detectados y el número total de ataques presentes en los datos:

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

(3)

Puntuación

Es la combinación de las métricas anteriores y nos permite averiguar cuán robusto es al ignorar instancias no detectadas. Se calcula como:

$$Puntuación = 2 * \frac{Precisión + Recall}{Precisión + Recall}$$

(4)

La ecuación general es:

$$F_\beta = (1 + \beta^2) * \frac{Precisión * Recall}{\beta^2 * Precisión + Recall}$$

(5)

7.2 Fase de Post-aprendizaje

En la fase de Post-aprendizaje se desarrollan los siguientes procesos en donde se manejan los datos en tiempo real. Destacaremos los tres primeros: Agregación de Tráfico, Cola y Extracción de Características que serán explicados con profundidad más adelante:

1. Agregación de tráfico: El objetivo de este primer paso reside en insertar datos recopilados de los sensores que estarán distribuídos en la red. No obstante, surge la problemática de encontrarnos en el caso de que se agreguen datos duplicados. Para ello, se comprobará que el *timestamp* del paquete recibido para anular la posibilidad de duplicidad. Si la marca de tiempo es igual, entonces se desconsiderará uno de los paquetes. Una vez terminado este proceso, se enviará la pila de paquetes a la cola.

2. Cola: Esta etapa encola los paquetes para facilitar la extracción de características.

3. Extracción de características: Se procederá a obtener las características de interés como se explica en el apartado de Selección

de Características. No obstante, y a diferencia de lo descrito hasta ahora, los atributos se recopilarán a tiempo real.

4. Clasificación del ataque: Nuestro modelo previamente entrenado y testeado clasifica el estado de nuestra red. Así pues, los estados pueden ser dos: estado de ataque y estado normal.

5. Módulo de decisión y aviso: Se trata de una unidad encargada de obtener un resultado de clasificación y elabora un paquete con intención de aviso —con bit 1, para estado anómalo de la red y bit 0, para estado normal de la red—, para ser enviado al agente IDS. El paquete se configura sobre el protocolo UDP, tendrá una carga útil y su tamaño será muy reducido. En su contenido encontramos el ID del nodo, el *timestamp* y el tipo de aviso. En caso de que el tipo de aviso sea normal, es decir, no se ha detectado ningún ataque, entonces se descartará el aviso al agente IDS. De forma contraria, si el aviso es de tipo anómalo, se enviará una trama de detección al agente. En ambos casos, se elaborará un registro en el repositorio local.

6. Repositorio local: Su función radica en almacenar los registros de todos los avisos, sea cual sea el resultado, con el fin de elaborar un plano de evidencia para distintos análisis de nuestra red.

7. API: Comprende el puente entre el usuario final y la base de datos local. En nuestro trabajo usaremos una API REST con las funcionalidades típicas de creación de datos —POST—, recuperación —GET—, modificación —GET— y eliminación de datos —DELETE—. Nuestra API estará formada por dos elementos: un primer elemento de lectura que obtendrá los datos del almacén de datos y los enviará al plano *fontend*. Y un segundo elemento para actualizar las informaciones incorrectas. Ésta última será usada por los administradores de la red. En adición, debemos destacar los tres niveles en los que estará formada nuestra API:

Endpoint: el cliente podrá elaborar solicitudes a la API REST para recuperar datos del almacén de datos.

Marco API REST: establece una unión entre el cliente *frontend* y el almacén de datos.

Almacén de datos: guarda las informaciones del nodo y su estado.

8. IDS o Agente IDS

El objetivo de este módulo es monitorear el tráfico de red ante posibles ataques. Cuando se detecte un ataque, el IDS difundirá un aviso en modo *broadcast* a todos los nodos de la red indicándoles el ID del nodo malicioso y su ruta. Con tal

propósito, los nodos tendrán constancia del elemento dañino de la red por lo que tomarán medidas para evitarlo y mitigar el ataque. En adición, el IDS reformulará la red creando rutas alternativas. De este modo, todo tráfico proveniente del nodo atacante será aislado e ignorado.

9. Exportación de Datos en Tiempo Real

Antes de estudiar el monitoreo de la red a tiempo real, desarrollaremos una herramienta de exportación y virtualización de los datos de nuestro sistema. En concreto, nos proporcionará información sobre el tráfico 6LoWPAN y RPL.

Debemos destacar que Cooja se basa en la *Java Native Interface* para la interfaz entre Java y el Sistema Operativo (SO), que en nuestro caso es, como sabemos, Contiki. De este modo, lograremos una emulación mediante sensores a través de MSPSim. [Webgrafía 10].

Lo que pretendemos, es implementar un recolector de datos a partir de los sensores en formato de texto simple CSV o MySQL. Para ello, usaremos un Marco de Exportación de Datos (DEF) que se puede integrar mediante la extensión para el plugin *Collect View* de Cooja. DEF está formado por 4 módulos: Módulo de Manejo de Datos (DHM), el Módulo de Visualización de Datos (DVM), el Módulo de Gestión de Base de Datos (DMM) y el Módulo de API (APIM).

9.1 Módulo de Manejo de Datos

El Módulo de Manejo de Datos es el encargado de recopilar todos aquellos datos de cada nodo o mota y guardarlos en una lista de tipo Array. El objetivo principal es almacenar los datos de los sensores para luego añadir la información en el Módulo de Visualización de Datos (DVM). Para tal propósito, lo que haremos será separar la recopilación en dos vertientes: una dedicada a la interfaz gráfica y otra para la representación de datos. Los nodos de la red junto con sus informaciones son guardados en un *ArrayLis*t. Prepararemos, por tanto, un pseudocódigo esquemático general cuyo fin es obtener una lista de informaciones de los nodos extrayendo así los datos más importantes. Dicha tabla se pasará al DVM.

Véase a continuación:

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_NODES 100 // Definimos el número máximo de nodos sensores

typedef struct {
    int data; // Datos del sensor
    int valid; // Indicador de validez del dato (1 si es válido, 0 si no)
} SensorNode;

typedef struct {
    SensorNode nodes[MAX_NODES]; // Lista de nodos sensores
    int count; // Contador de nodos válidos
} SensorDataList;

// Agregamos un nodo sensor a la lista de datos del sensor
void addSensorData(SensorDataList *list, SensorNode node) {
    if (list->count < MAX_NODES) {
        list->nodes[list->count++] = node;
    }
}

//Agregamos y actualizamos los datos de los sensores
SensorDataList aggregateSensorData(SensorNode data[], int n) {
    SensorDataList list = {{0}, 0};

    // Inicializamos la lista de datos del sensor con el primer elemento
    if (data[0].valid) {
        addSensorData(&list, data[0]);
    }

    for (int i = 1; i < n; i++) {
        if (data[i].valid) {
            addSensorData(&list, data[i]);
        } else {
            printf("Datos Null\n");
        }
    }
    return list;
}
```

Figura 44: Código general manejo datos entrantes al sensor

Entonces, a grandes rasgos tenemos que, por una parte, *addSensorData* añade un nodo sensor a la lista de datos si la lista no está llena y, por otra parte, *aggregateSensorData* agrega y actualiza los datos de los sensores; inicializa la lista con el primer nodo válido y luego recorre el resto de los nodos, agregando los válidos y manejando los nulos.

9.2 Módulo de Visualización de Datos

Este módulo muestra en una interfaz gráfica los datos permitiéndole así al administrador de la red monitorear el rendimiento de la misma, así como seleccionar, filtrar, buscar y configurar la pila de datos. En adición, este módulo permite al administrador exportar los datos en diferentes formatos como, por ejemplo, CSV. En definitiva, se trata de aportar una interfaz agradable para el manejo de información.

9.3 Módulo de Gestión de Base de Datos

Trata la interfaz de base de datos. Por conveniencia, usaremos CSV y MySQL dado que son formatos extensamente usados. Desarrolla además una clase de gestión de información para configurar los usuarios y actualizar sus tablas de datos. Para ello, se puede emplear el controlador JDBC de Java [Recursos 12] permitiéndonos conectar el Marco de Exportación de Datos con MySQL. Así, el uso que le podemos dar es conectarnos a una interfaz web para monitorear los nodos, su estado y función, entre otros.

9.4 Módulo API

Utilizaremos una API REST cuyo objetivo será obtener las informaciones de la base de datos que deseemos y convertirlas a un flujo de datos adecuado para otras plataformas deseadas. Dicha API se dividirá en dos clases: la primera, *Clase BaseDeDatos()* que conecta la base de datos que contine información de los nodos, y la segunda *ObtenerData().* Esta última se encarga de obtener informaciones concretas del nodo de la base de datos. Por otro lado, también convertirá las informaciones en formato JSON que será de utilidad para el manejo en otras aplicaciones.

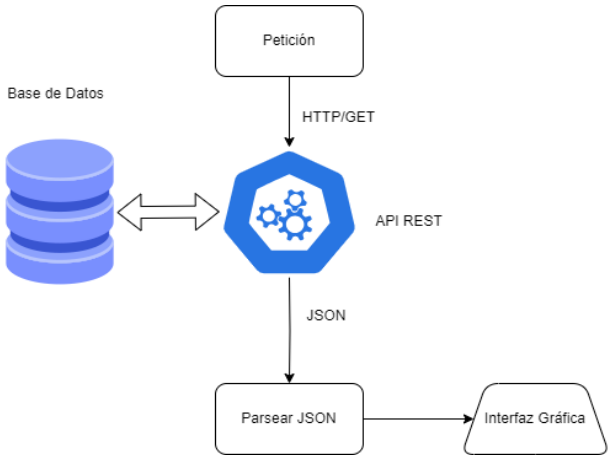


Figura 45: Marco de Exportación de Datos API REST

En la Figura 46 se muestra una solicitud HTTP a la API REST. Se inicia una conexión a la base de datos enviándolos al plano *front-end.* En este punto, los datos obtenidos se codifican, como hemos dicho en formato JSON conseguidos por la API DEF. Se muestra la lectura de la potencia de la mota junto con el ID.

```
[{"NodeID":"5","PowerTrace":"47680"}]
```

Figura 46: Salida JSON

10. Colección de Datos a Tiempo Real

A continuación, presentamos el esquema general de nuestra red para la generación de datos y monitoreo en tiempo real que enviará el tráfico del nodo malicioso a la unidad IDS instalada en el router 6LoWPAN.

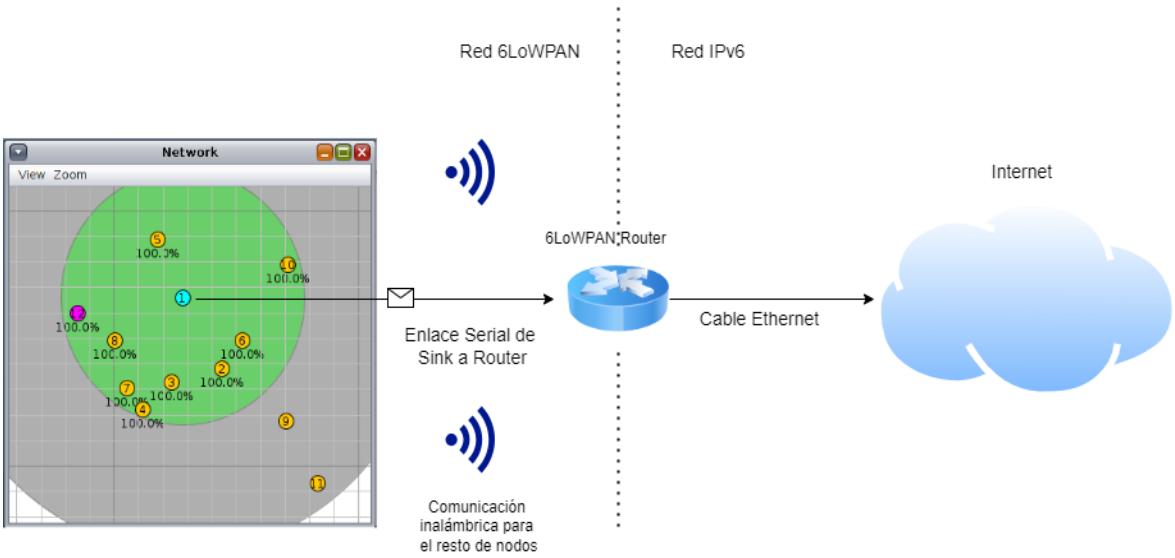


Figura 47: Esquema físico de ubicación de red e IDS

Como vemos, debemos considerar una nueva arquitectura de red para ajustar nuestra captura de datos lo máximo que podamos al mundo real. Al ser una red basada en RPL, contaremos con un router 6LoWPAN cuya función será intercambiar la información entre los nodos e Internet. Dichos elementos se conectarán al router mediante conexiones inalámbricas. Por otro lado, dispondremos de *sniffers* distribuidos en nuestra red que capturarán los datos que serán retransmitidos a los siguientes 4 componentes: el Medio de Captura, la Agregación de Datos, la Unidad de Encolado y la Unidad de Extracción de Características. Véase en la Figura 48:

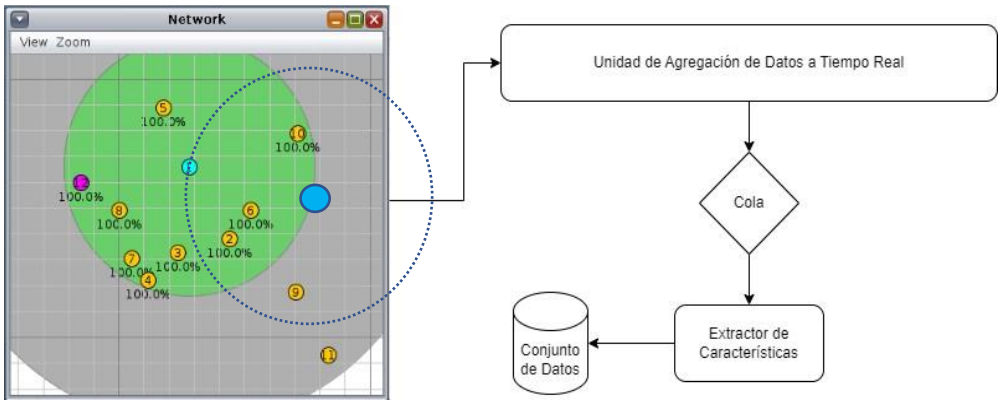


Figura 48: Módulo Colección de Datos

En los siguientes puntos se describe el esquema anterior.

Captura de Datos

La captura de datos en bruto se efectúa mediante sondas (punto color azul) distribuidas en nuestra red que recopilan datos que son enviados a la Unidad de Agregación de Datos a Tiempo Real. Esto lo podemos hacer mediante el modelo Sensniff que nos permitirá llevar a cabo esa tarea. Sensiff [Recursos 6] es un *firmware* equipado con las bibliotecas Libpcap [Recursos 7]que captura tráfico de la Capa de Enlace.

Aprovechando los conocimientos adquiridos sobre algoritmos de la asignatura Prácticas de Programación (75.555) cursada durante el grado, desarrollamos el siguiente proceso en lenguaje algorítmico para tal propósito:

```
{Resultado: paquete}
paquete = paquete entrante;

{Minetras sea distinto de cero}
while paquetes.size != 0 do
     tiempo = paquete[0].timestamp - paquete.timestamp;
     if paquete.size =¡ 40 then
          envia paquete.payload to cola;
     else
          continua;
     end
end
```

Figura 49: Esquema algorítmico de colección de datos

El algoritmo anterior capta los datos en bruto para ser enviados a la unidad de agregación de datos.

Agregación de Datos

Este módulo inserta los datos de las fuentes asegurando la integridad de los mismos, pues no nos conviene obtener datos duplicados. En este plano observamos el número de nodos conectados y los agregamos a una tabla. Seguidamente recorrerá cada nodo verificando la integridad de los datos y su marca de tiempo o *timestamp.* Si una información se ve duplicada, se identificará mediante la comparación de los atributos de cada paquete, es decir, ID, IP origen, IP destino y marca de tiempo. Los datos dobles se eliminarán.

Cola

La cola es el órgano de nuestra estructura que recibe los paquetes.

Unidad de Extracción de Características

Se trata de una unidad más importante puesto que agrupa la información de los paquetes en tres capas —Capa Física, Capa de Red y Capa de Aplicación— y extrae sus correspondientes datos. Para ello, usaremos la ya mencionada biblioteca Libpcab.

- Características de Capa Física

  Se agruparán los datos como la señal o el rango de transmisión. El DCM será el encargado de dicha tarea.

- Características de Capa de Red

  Se extraen datos de los protocolos 6LoWPAN y RPL, número de vecinos, número de mensajes DIO, número de mensajes DIS, entre otros.

- Características de Capa de Aplicación

  Se extraen datos del sensor en la Capa de Aplicación. Siguiendo nuestro ejemplo de aplicación para un sistema de riego inteligente, de ahí obtendremos datos sobre la temperatura, humedad o energía del nodo— las informaciones de esta capa podrán variar en función de la función de la red—. La energía de cada nodo, por ejemplo, nos permitirá estudiar cómo afecta el ataque a nuestra red.

Finalmente, debemos considerar clasificar los datos por etiquetas con el fin de averiguar rápidamente en qué segmento se originó el ataque y aislar el nodo que está infectando la red.

11. Configuración del modelo de aprendizaje

En el Capítulo de Arquitectura de Detección planteamos un primer avance de la estructura de nuestro modelo. Basándonos en lo anterior, configuraremos un modelo de aprendizaje automático empleando el conjunto de datos obtenidos hasta ahora. Dividiremos la construcción de este modelo en tres fases: introduciremos la teoría matemática de dos algoritmos de aprendizaje automático —Support Vector Machines (SVM) y Random Forest—, elaboraremos un estudio intenso de nuestros modelos de aprendizaje bajo nuestro escenario de ataque y finalmente concluiremos en que el método de aprendizaje automático SVM para ataques DDoS y DoS es el más adecuado.

11.1 Máquina de Vectores de Soporte (SVM)

Es un algoritmo de aprendizaje automático utilizado en regresión y clasificación al tomar el vector de entrenamiento entrante distinguiendo los datos en clases basadas en las características n-dimensionales. El primer concepto que debemos entender es el hiperplano que divide los datos en distintas clases dependiendo de sus características. Así mismo, debemos considerar la existencia de múltiples hiperplanos en los conjuntos de más de una dimensión. En consecuencia, el objetivo radica en adquirir el plano más adecuado. Si queremos agrupar exitosamente los puntos de datos, entonces el algoritmo deberá establecer la distancia máxima entre hiperplano y el punto de datos usando la función de pérdida de bisagra como se muestra en la siguiente:

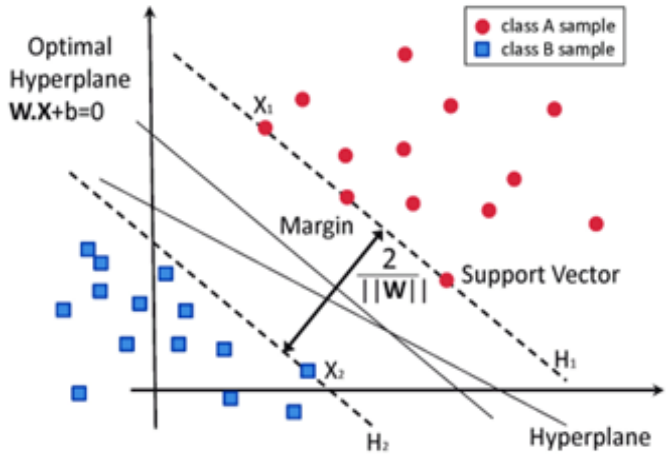


Figura 50: Diagrama SVM

La ecuación que representa el hiperplano es:

$$W * X + b = 0$$

(6)

b es el término de sesgo, **X** es el vector de puntos de datos de entrada y **W** es el peso variable. Entonces, para cada punto de datos encontramos dos vertientes: el valor es parte de la clase (+), es decir, obtenemos 1 y, por otra parte, el punto de datos es parte de la clase (−), es decir, obtenemos −1.

$$\begin{cases} W * X + b = 1 \\ W * X + b = -1 \end{cases}$$

(7)

Supongamos ahora que tenemos un conjunto de datos tipo $\mathbf{x_i} \in R^d$, con $\mathbf{i = 1,,,,,t}$ donde **x** representa el punto de datos y los superpuntos **(i)** indican la aparición de esa instancia, considerando que $\mathbf{x_i}$ forma parte de cualquiera de las dos clases $\mathbf{y_i} \in \{1,-1\}$. Por tanto, estableceremos la clase 1 como el tráfico normal y −1 como excepciones o anomalías de tráfico. De acuerdo con las ecuaciones 6 y 7, escribimos una nueva ecuación

$$\begin{cases} W^T * X_i + b \geq 1 \, para \, todo \, X_i \in Normal \\ W^T * X_i + b \leq -1 \, para \, todo \, X_i \in Normal \end{cases}$$

(8)

En caso de que el conjunto pudiese ser separado linealmente, se expresaría mediate:

$$y_i \, (W^T * X_i + b) \geq 1 \, para \, todo \, i = 1, \dots L$$

(9)

La línea de puntos que separa nuestro hiperplano se conoce como margen y compromete la distancia entre el vector de soporte de las clases. Ésta se calcula como $\frac{2}{||W||}$ que se puede escribir minimizando $||W||^2$. En definitiva, SVM tiene como objetivo buscar el hiperplano más adecuado maximizando el margen entre hiperplanos y clases. Esto lo conseguimos con las ecuaciones (9) y (10):

$$M(W) = \frac{1}{2} \, ||W||^2$$

(10)

La ecuación 9 deviene un problema de optimización que se usa mejore n conjuntos que tienen una separación entre clases que pueden separarse linealmente. No obstante, nos encontramos en un caso en que los conjuntos complejos de datos no pueden separarse linealmente. En consecuencia, se usa el Problema Lagrange [Webgrafía 16] junto con el Problema Dual de Wolf [Webgrafía 3]:

$$\mathrm{L}(W, b, \alpha) = \frac{1}{2} \, W \cdot W - \sum_{i=1}^{m} \alpha_i [y_i (W \cdot X + b) - 1]$$

(11)

Se pretende maximizar el multiplicador de Lagrange α para cada instancia de X_i. Los puntos anteriores, sin embargo, pueden llevar a problemáticas de ruido en nuestro conjunto de datos. Para solucionarlo, usamos la restricción con la variable de holgura:

$$y_i (W \cdot X_i + b) \geq 1 - \zeta_i, \, \mathrm{i} = 1 \dots \mathrm{m}$$

(12)

Teniendo en cuenta lo anterior, nos queda:

$$\min W, b, \zeta \, \frac{1}{2} ||W||^2 + \sum_{i=1}^{m} \zeta_i$$

(13)

$$y_i (W \cdot X_i + b) \geq 1 - \zeta_i, i = 1 \dots m$$

(14)

En este punto, nos surge el problema de elegir un valor demasiado grande para la holgura, es por eso que introducimos el parámetro C. De este modo, un valor reducido para C enfatizará la importancia de ζ. Así, pues, las nuevas ecuaciones nos quedan:

$$\min W, b, \zeta \frac{1}{2} ||W||^2 + C \sum_{i=1}^{m} \zeta_i$$

(15)

$$y_i(W \cdot X_i + b) \geq 1 - \zeta_i, \zeta_i \geq 0 \, i = 1 \ldots m$$

(16)

Finalmente, sustituyendo (15) mediante el problema dual en Lagrangiana obtenemos las siguientes funciones que constituyen los objetivos de SVM a resolver con el fin de trazar el hiperplano y separar los puntos de datos:

$$\max \alpha \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

(17)

$$\alpha_i \leq C, i = 1 \ldots m, \sum_{i=1}^{m} a_i y_i = 0$$

(18)

Veamos a continuación qué sucede si los datos sin ruido no son linealmente divisibles. Los datos, en el mundo real, cuentan en la mayoría con ruido o valores faltantes o, en su defecto, están en formatos inadecuados que no se pueden utilizar directamente para modelos de aprendizaje automático. Para ello, se usa el truco de Kernel que aplicado a las ecuaciones anteriores obtenemos:

$$\max \alpha \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(X_i \cdot X_j)$$

(19)

$$\alpha \geq 0, \mathrm{i} = 1 \ldots \mathrm{m}, \sum_{i=1}^{m} \alpha_i y_i = 0$$

(20)

Para SVM podemos usar distintos Kernels como el lineal, para clasificaciones de texto, el polinominal, para mayor fiabilidad o el Kernel en función de base radial definido (RBF) como $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||2)$. Éste último será el más adecuado para nuestro caso.

11.2 Bosques Aleatorios

Un árbol de decisión se define como un modelo de aprendizaje dedicado a la clasificación y predicción en el ámbito de aprendizaje automático y está basado en los árboles de decisión. En este, cada punto se conoce como nodo. Empezando por el primero, establecemos el nodo raíz, mientras que los demás se le darán el nombre de hojas. El árbol irá tomando un seguido de decisiones binarias —certero o falso— mientras cada nodo está conectado a otros dos nodos. El conjunto de datos, por lo tanto, se dividirá en clases en función de sus atributos mediante el método de división. En nuestro caso, usaremos el bosque aleatorio o también llamado *random forest* con el método de División de Impureza de Gini [Webgrafía 4] y de antropía. Los bosques aleatorios se definen como árboles de decisión que hace aleatorio la configuración de un árbol en el "bosque" aleatorizando las muestras de entrenamiento y selección de subconjuntos en la generación de un árbol.

Siguiendo con el hilo anterior, presentamos a continuación el método de División de Impureza de Gini [Webgrafía 4]:

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

(21)

12. Entrenamiento y validación.

El propósito del presente trabajo es elaborar una simulación de Ataque de Denegación de Servicio con el fin de poder aplicar una solución para la detección de dicha ofensiva, con todo, sirviéndonos de los datos obtenidos mediante el IDS desarrollado aplicado sobre nuestro escenario ficticio. No obstante, y desafortunadamente, los recursos y herramientas para la captura de datos son limitados y pueden presentar deficiencias en la precisión del entreno de nuestro modelo. En consecuencia, usaremos un conjunto de datos o *dataset* público llamado IoTR – DS [Recursos 8], dedicado a redes IoT especializado en RPL y 6LoWPAN. Este *dataset* recoge algunas características relacionadas con el tráfico normal y de ataque IoT. Concretamente, cuenta con 4 *subsets* que representan distintos escenarios etiquetados en función de sus características. El primero representa el estado normal de la red, el segundo representa un ataque DIS, los dos últimos abordan un ataque de rango y otro de *wormhole*. Dada la naturaleza de la simulación estudiada en los capítulos anteriores, nos centraremos

en el *subset* dedicado al ataque *DIS flood* y *RPLNormal* [Recursos 9 y 17] IoTR-DS.
Por otra parte, debemos considerar que los metadatos comprenden escenarios con una red de cien nodos y una duración de 83 minutos. Si bien es cierto que en simulaciones anteriores hemos considerado un total de doce nodos y sesenta minutos debido a las limitaciones de nuestros recursos de simulación y captura, podemos aplicar este *dataset* público para entreno de nuestro modelo, pues el ataque que se desarrolla es el mismo y, en adición, al contar con más cantidad de datos, obtendremos una mayor precisión en la formación del algoritmo. Si observamos el conjunto de datos, vemos que los atributos son los que esperábamos en simulaciones anteriores –número de secuencia del paquete, tiempo de recepción y envío, dirección IP de origen, ID del nodo, etc–.

Entonces, los *subsets* obtenidos del escenario se divide en:

Dataset	Tráfico normal	Ataque	Total
Ataque DIS	60422	16353	76775
Estado normal	93000	0	93000

Tabla 8: Subsets obtenidos.

A continuación, se introducen y estudian los procesos de entrenamiento de los algoritmos de aprendizaje automático anteriormente descritos. Los métodos son entrenados mediante Scikit-learn, una biblioteca de aprendizaje automático de software libre para Python [Recursos 10].

12.1 Entrenamiento

Normalización

No obstante, antes de proceder con el entreno debemos normalizar y codificar las etiquetas. En cuanto al primer paso, se trata de escalar los datos con la intención de ajustarlos en un rango establecido. Para ello, transferiremos los valores que aporta el conjunto de datos a un rango entre 0 y 1. Para tal cometido, usamos el escalador estándar de Scikit-learn: z = (x − u)/s, o también, la fórmula:

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

(22)

Siendo x_{norm} el valor normalizado, x el valor original, $min(x)$ el valor mínimo del conjunto de datos y $max(x)$ el valor máximo del conjunto de datos.

Etiquetado

Por otra parte, debemos codificar las etiquetas. Las variables categóricas pasan a ser números a fin de que sea de mejor interpretación para los algoritmos, pues éstos se comportan mejor ante datos numéricos. La codificación de etiquetas también se elaborará con Scikit-learn.

Una vez entrenado el algoritmo, en la fase de verificación dividiremos nuestro conjunto de datos en diferentes subconjuntos. Dividiremos el conjunto de datos total en un 70% para el entrenamiento -mediante el método *train_test_split*- y un 30% para la prueba o testeo de nuestro algoritmo.

12.2 Validación

Validación Cruzada

Anteriormente, hemos indicado que dividimos el *dataset* en una proporción de 70% - 30%, no obstante, debemos tener en cuenta que, a pesar de ello, los subsets puede que no tengan la misma distribución, es decir, puede que el subset de entrenamiento esté formado por datos ligeramente diferentes al de validación y prueba, lo cual puede afectar a nuestro entrenamiento. En adición, si ajustamos los parámetros e hiperparámetros siempre con los mismos sets podemos caer en el error de que nuesto modelo se sobreajuste, es decir, memorice de alguna manera los datos de entrenamiento y testeo y genere errores de predicción.

Con el fin de evitar este sesgo, utilizaremos el algoritmo k-fold para validar de forma cruzada los conjuntos de datos. Es decir, k-fold creará varios subconjuntos de datos de entrenamiento y prueba alternándolos entre ellos. Nosotros utilizaremos 6 subconjuntos.

13. Código de entreno

Finalmente, pasamos al entreno de nuestros algoritmos. Si bien es cierto que el grado de Ingeniería de Tecnologías y Servicios de Telecomunicación no recoge la enseñanza de porgramación en Python, después de un estudio profundo de los recursos [13]-[16], presentamos a continuación un pseudocódigo que realizará las tareas anteriormente descritas.

```python
# Importamos las bibliotecas necesarias
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

# Cargamos el dataset
df = pd.read_csv('DISAttack-5000s-updated2.csv')

# Normalizamos los datos con StandardScaler
scaler = preprocessing.StandardScaler()
df_scaled = scaler.fit_transform(df)
df = pd.DataFrame(df_scaled)

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, :-1], df.iloc[:, -1], test_size=0.3, random_state=42)

# Entrenamos el modelo SVM
clf_svm = svm.SVC()
clf_svm.fit(X_train, y_train)
```

```python
# Realizamos la validación cruzada para el modelo SVM
scores_svm = cross_val_score(clf_svm, X_train, y_train, cv=6)

# Generamos la matriz de confusión para el modelo SVM
y_pred_svm = clf_svm.predict(X_test)
confusion_matrix_svm = confusion_matrix(y_test, y_pred_svm)

# Entrenamos el modelo Random Forest
clf_rf = RandomForestClassifier()
clf_rf.fit(X_train, y_train)

# Realizamos la validación cruzada para el modelo Random Forest
scores_rf = cross_val_score(clf_rf, X_train, y_train, cv=6)

# Generamos la matriz de confusión para el modelo Random Forest
y_pred_rf = clf_rf.predict(X_test)
confusion_matrix_rf = confusion_matrix(y_test, y_pred_rf)

print("Matriz de confusión SVM:", confusion_matrix_svm)
print("Matriz de confusión Random Forest:", confusion_matrix_rf)
```

Figura 51: Código de entreno

Como se observa, se elabora un flujo completo de procesamiento y análisis del *dataset* utilizando técnicas de aprendizaje automático.

En primer lugar, se importan las bibliotecas necesarias para la manipulación de datos, así como de la división de conjuntos, entrenamiento y evolución. Cargaremos el conjunto de datos mediante *df = pd.read_csv('DISAttack-5000s-updated2.csv'),* una vez cargado, se normalizan los mismos mediante *StandardScaler* de scikit-learn. También se dividen el *dataset* en sebsets de entrenamiento (70%) y prueba (30%) usando la función *train_test_split.* Seguidamente, nos dispondremos a entrenar los datos tanto de SVM como de RF. No obstante, para evitar el sobreajuste y que los modelos se aprendan de memoria los datos, se realiza la validación cruzada con 6 particiones (cv=6) para evaluar el rendimiento del modelo SVM y RF en diferentes subconjuntos del conjunto de entrenamiento, véase: *scores_rf = cross_val_score(clf_rf, X_train, y_train, cv=6)* y *scores_svm = cross_val_score(clf_svm, X_train, y_train, cv=6).* Finalmente, generaremos e imprimiremos las matrices de confusión obtenidas.

Capítulo V. Análisis de resultados

Como se indica en el apartado de Configuración de Modelo, se debe usar el Kernel RBF en SVM a consecuencia de la imposibilidad de separar los datos linealment, además, debemos ajustar los hiperparámetros para tener una buena precisión de predicción. Por otro lado, SVM no escala escala bien con datasets muy grandes, ya que el entrenamiento puede ser lento y consumir mucha memoria. En otras palabras, es menos interpretable que Random Forest, ya que las decisiones se basan en funciones de kernel que no son fácilmente comprensibles.

A consecuencia de lo anterior, prevemos unos mejores resultados para RF sobre SVM.

1. Evaluación SVM

Observemos a continuación la matriz de confusión obtenida para SVM:

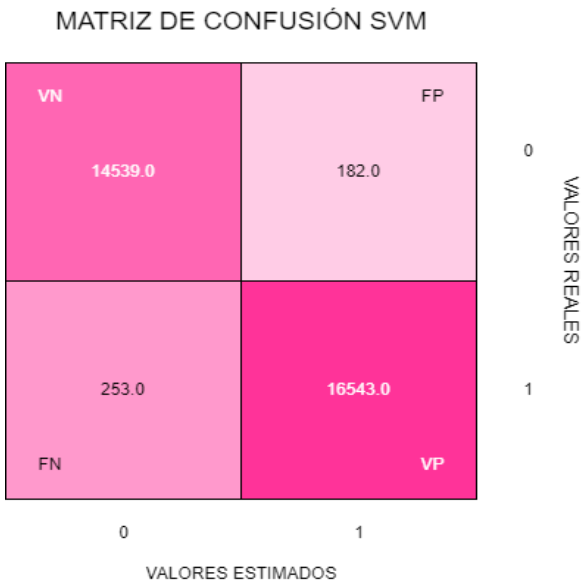


Figura 52: Matriz de confusión SVM

Como comentamos en apartados anteriores, la Matriz de Confusión nos ayuda a entender cuán bien funciona la predicción de un modelo de clasificación; recordemos a continuación su funcionamiento. Agrupa las predicciones según Verdaderos Positivos (VP), Verdaderos Negativos (VN), Falsos positivos (FP) y Falsos Negativos (FN) en una escala de rosados. En adición, observaremos en los márgenes el valor 1 (positivo), 0 (negativo), valores reales y valores estimados [Webgrafía 19]. De esta manera, y a modo de ejemplo, podemos observar que en el punto 01 (FP), obtenemos un resultado de 182.0. Estas métricas por sí solas no son útiles, sino que nos sirven para calcular los siguientes mediante las fórmulas ya descritas en el apartado 7.1.3.

Aplicando las fórmulas ya descritas, obtenemos Exactitud = 98.6%, Precisión = 98.2%, Recall = 98.7%, Puntuación (también llamado F-Measure) = 98.5%

2. Evaluación Bosque Aleatorio (Random Forest)

Observemos ahora la matriz de confusión obtenida para RF:

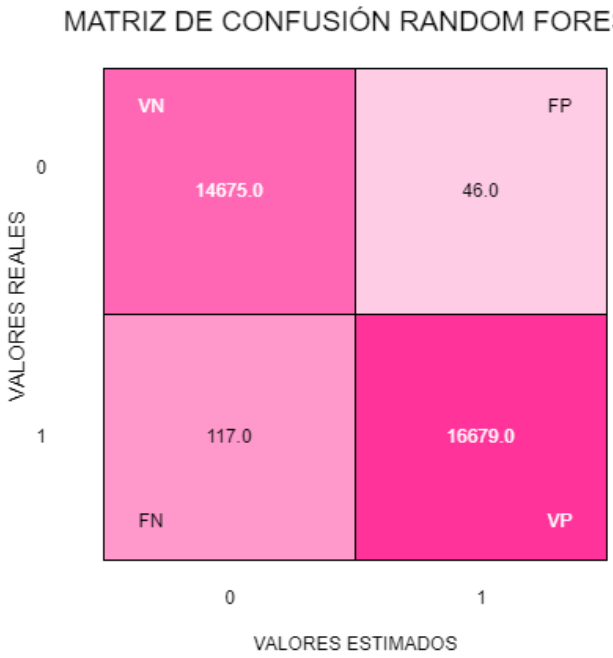


Figura 53: Matriz de confusión Random Forest

Aplicando las fórmulasconocidas, obtenemos Exactitud = 99.4%, Precisión = 99.2%, Recall = 99.6%, Puntuación (también llamado F-Measure) = 99.4%. Agrupando los datos obtenidos de las dos evaluaciones obtenemos las siguientes tablas en donde compararemos los resultados:

Clasificador	Exactitud
SVM	98.6%
Random Forest	99.4%

Tabla 9: Comparaciones exactitud.

Clasificador	Precisión
SVM	98.2%
Random Forest	99.2%

Tabla 10: Comparaciones Precisión.

Clasificador	Recall
SVM	98.7%
Random Forest	99.6%

Tabla 11: Comparaciones Recall.

Clasificador	F-Measure
SVM	98.5%
Random Forest	99.4%

Tabla 12: Comparaciones F-Measure.

3. Estudio final

A continuación, elaboraremos un estudio de los resultados obtenidos.

En la Tabla 9 se demuestra la exactitud de nuestros dos modelos de aprendizaje automático (ML): SVM y Random Forest (RF). Entendemos con la exactitud el porcentaje de predicciones acertadas realizadas en comparación con el valor real de la etiqueta. La tasa de exactitud para SVM fue del 98.6% mientras que en Random Forest fue de 99.4%. Según los datos recopilados, es el Random Forest el que supera a SVM en términos de precisión. Esto se debe a que RF, como sabemos, utiliza múltiples árboles de decisión, es decir, comprende menos errores de clasificación.

La medida de precisión se comprende como el porcentaje que muestra qué proporción de los objetos reconocidos son pronósticos acertados. En la Tabla 10 observamos que SVM obtuvo un porcentaje de 98.2% y Random Forest alcanzó un 99.2% respectivamente; el clasificador RF fue el más preciso.

Por otra parte, en la Tabla 11 estudiamos el *Recall* o sensibilidad. Ésta se trata de una medida muy utilizada en el aprendizaje automático con el fin de evaluar modelos de clasificación binaria, por lo que indica el número de casos verdaderamente positivos que nuestro modelo categorizó correctamente como tal. Los valores de *Racall* obtenidos son de 98.7% en SVM y de 99.6% en Random Forest. De nuevo, los valores de sensibilidad exponen una cierta superioridad en RF sobre SVM con una diferencia de 0.9%.

Finalmente, en la Tabla 12 se compara el desempeño o también llamado *F-Measure* (Medida F en castellano). Se trata de una métrica beneficiosa cuando hay una distribución desigual de clasificaciones positivas y negativas. Los porcentajes obtenidos son de 98.5% en SVM y 99.4% en RF, lo cual pone en primer lugar a RF frente a SVM.

Como se puede observar en nuestro análisis, Random Forest se muestra superior en exactitud, precisión, *recall* y F-Measure, presentándose así, como el modelo de aprendizaje automático más adecuado para implementar en nuestro Sistema de Detección de Intrusos para detectar Ataques de Denegación de Servicio en entornos IoT.

Capítulo VI. Contribuciones y Valoración económica

El presente trabajo desarrolla un marco basado en el aprendizaje automático para detectar Ataques de Denegación de Servicio en entornos IoT. Las dos principales contribuciones del trabajo se exponen a continuación:

1. Marco de protección usando ML

La elaboración de soluciones en redes del Internet de las Cosas supone un reto difícil, pues se trata de entornos cambiantes con recursos limitados. Para tal propósto, hemos estructurado un plano que abarque dichos inconvenientes

Recolección de datos: En la fase de Post-Aprendizaje se adquieren los datos a tiempo real permitiéndonos el monitoreo constante de una red tan cambiante como IoT.

Módulo de detección: Constituye el punto clave en cuanto a detección y análisis. Éste, se implementa en un nodo de borde o *router* frontera.

API: Nos permite el almacenamiento de los resultados para ser consultados y ordenados en un repositorio local que puede ser consultado por otras aplicaciones de terceros.

2. Metodología Inteligente

El Capítulo IV expone con rigor matemático los procesos algorítmicos de Máquinas de Vector de Soporte y Random Forest. Dichos modelos de aprendizaje automático se entrenaron y evaluaron para detectar ofensivas DoS. En adición, se cuidaron especialmente los procesos con el fin de obtener resultados sólidos y aislar las conclusiones sesgadas. Hasta ahora, ninguna investigacióin previa indaga en el uso de algoritmos de aprendizaje automático para la detección de ataques DIS-flood en redes IoT.

3. Valoración económica

En cuanto al presupuesto del proyecto, valoraremos principalmente dos aspectos: el coste humano y el coste material.

Concepto	Descripción	Unidades	Precio	Total
Diseño, elaboración, imlpementación y mantenimiento del IDS	Funciones Ingeniero de Redes y Sistemas de Telecomunicación	420h	50€/h	21000€
			Total:	21000€

Tabla 13: Coste Humano.

Concepto	Descripción	Unidades	Precio	Total
Motas S_1	Motas Sky de la simulación 1	8	70€	560€
Mota S_2	Motas Z1 de la simulación 2	12	100€	1200€
Router fronetera	Router 6LoWPAN	1	76€	76€
Enlaces	Enlaces por cable router (Serial Link y Ethernet	2	50€ 10€	60€
Sensores	Sensores CC2420	2	119,45€	239€
			Total:	2135€

Tabla 14: Coste Material.

Finalmente, obtenemos un coste total de **23.135€**

Capítulo VII. Conclusión

Los entornos IoT crecen rápidamente abarcando áreas y dominios hasta ahora desconsiderados. En consecuencia, esta proliferación demanda una adaptación a diferentes sectores que acompañan consigo muchos desafíos de seguridad. El más común en este grupo son los Ataques de Denegación de Servicio que merma las capacidades de la red y afecta directamente a los recursos provocando pérdidas económicas y de calidad en los servicios. Este trabajo presenta una posible solución a la detección y mitigación de dichas amenazas utilizando dos modelos de aprendizaje automático: SVM y Random Forest. En el grupo de ataque DDoS, se simula sobre una red IoT basada en RPL el ataque de inundación DIS-Flood, el cual se aborda en la fase de Post-Aprendizaje mediante los dos clasificadores anteriores. Después del entreno y testeo de los modelos, se observa superioridad efectiva en Random Forest para detectar dicho ataque. Para dar respuesta a tales ofensivas, se desarrolla un Sistema de Detección de Intrusos (IDS) en una red basada en Contiki OS en donde se discuten diferentes

implementaciones sobre algoritmos de recolección y agregación de datos en tiempo real en la red Io, todo, junto con un módulo de extracción de características. Además, se estudia la capacidad del IDS de reducir el consumo de energía de los nodos al mitigar el ataque aislando el nodo atacante.

Se proponía en las bases de este trabajo el objetivo de abrazar los trs pilares de la ciberseguridad: confidencialidad —datos accesibles por los usuarios oportunos, integridad —precisión en la información—, y disponibilidad —información y servicios disponibles el máximo de tiempo posible—.

Aseguramos, pues, después del trabajo realizado, el alcance de los mismos; solo los usuarios legítimos de la red acceden a los datos y servicios, ciñéndonos así a la confidencialidad; la red se mantiene íntegra, es decir, los datos tampoco no se alteran o corrompen; y finalmente, se disuaden los ataques manteniendo firme la red, acogiendonos a la disponibilidad.

1. Limitaciones y líneas futuras de trabajo

Aunque se ha realizado un trabajo que estudia diferentes aspectos de nuestros objetivos, se han presentado diferentes problemáticas debido a la limitación de los recursos, el alcance de los mismo, tiempos y criterios condicionantes. Así pues, todavía queda margen para fortalecer el sistema propuesto. Debemos recordar que los entornos IoT son cambiantes y pueden abordar todo tipo de arquitecturas. En nuestro caso, hemos realizado simulaciones con un número de nodos considerablemente bajo para facilitar la exposición, sin embargo, en la vida real, las redes están compuestas de muchos dispositivos y condicionantes. Además, no se desarrolla profundamente la implementación de un IDS, sino que se confecciona la arquitectura y diseño del mismo, pudiendo aplicar todavía más módulos y funcionalidades.

Asimismo, planteo también al lector el esfuerzo que conlleva realizar este trabajo de la manera más rigurosa posible, pues como recalco anteriormente, se han introducido dificultades debido a las limitaciones. La más desafortunada, quizá, se comprende en el Capítulo IV en donde se implementa el código de entreno de nuestros modelos. Al proponerse un proyecto tan extenso es inevitable toparse con la relación entre complejas y distintas temáticas y, por ende, la deriva —de dificultosa previsión— a nuevos campos desconocidos para el investigador. Así pues, dicho capítulo, el Capítulo IV, en especial la creación del codigo Python, se ha desarrollado en ese marco, dado que el Grado de Ingeniería de Tecnologías y Servicios de Telecomunicación no recoge la enseñanza de ese tipo de lenguaje de programación —ni su dominio merece un aprendizaje rápido—, redundando todo, en consecuencia, en un resultado no tan exacto como se podría haber dado en caso contrario. A pesar de ello, se han sorteado estos bloqueos con una revisión y esfuerzo exhaustivo dedicado a entender el arte de este lenguaje de codificación.

Con el fin de mejorar el proyecto, y con aspiraciones a las líneas futuras de trabajo, se propone explorar otros modelos de aprendizaje automático; recordemos que se han evaluado dos, pero hay otros como, por ejemplo, el

conocido XGBoost. También se analiza un solo ataque, sin embargo, encontramos Wormhole, Blackhole, Selective Forwarding Attacks, entre otros.

Se propone desarrollar además un esquema que permita implementar IDS a todos los nodos eliminando la necesidad de nodos *sniffing* externos. En el futuro, se abre una nueva posibilidad de crear un IDS que no dependa del enrutador 6RB. Esto puede lograrse con la computación FOG en cuanto esté lo suficientemente desarrollada y adaptada para usarse sobre dispositivos IoT.

2. Reflexión final

A título personal, y después definalizar los objetivos, contemplo un proyecto que ahonda en aspectos modernos del campo de la Ciberseguridad. Reflexionando sobre los inicios de este trabajo, se me ocurren algunos puntos de mejora para estudios futuros. Si bien es cierto que concluyo exitosamente este trabajo —por lo menos a título personal—, sí observo algunos errores que cometí en los inicios de la misma al esquematizar una planificación mediocre que fue revisada y mejorada en los inicios de la PEC 2. En adición, se me presentaron distintas problemáticas para entrenar y probar los modelos de aprendizaje, pues desconsideré en la planificación que iba a operar con recursos limitados, por lo que tuve que adaptarme a recursos públicos y gratuitos disponibles para la gran mayoría —*softwares* públicos, potencia de *hardware,* etcétera—. En la fase de simulación, por ejemplo, se barajó la posibilidad de usar el programa *NetSim Simulator*, sin embargo, dicha propuesta fue descartada por ser una opción con coste de licencia. No obstante, considero haber sorteado dichas acotaciones u obstáculos elaborando un trabajo que cumple con los objetivos propuestos.

Finalmente, agradezco el apoyo del Prof. Fernando Pérez López por acompañarme y estar pendiente de las cuestiones y dudas que le he planteado durante estos últimos meses de estudio.

Glosario de Abreviaturas

IoT:	Internet Of Things
DoS:	Ataques de Denegación de Servicio
DDoS:	Ataque de Denegación de Servicio Distribuidos
IA:	Inteligencia Artificial
DL:	Aprendizaje Profundo
ML:	Aprendizaje Automático
SVM:	Support Vector Machine
RF:	Ranfom Forest
IDS:	Sistema de Detección de Intrusos
TCP:	Protocolo de Control de Transmisión
UDP:	Protocolo de datagramas de usuario
ICMP:	Protocolo de control de mensajes de Internet
HTTP:	Protocolo de transferencia de hipertexto
RPL:	Protocolo de Enrutamiento para Redes de Baja Potencia y Alta Pérdida
DODAG:	Grafo Acíclico Dirigido Orientado al Destino
DIS:	DODAG Information Solicitation
DIO:	DODAG Information Object
DAO:	Destination Adversiment Object
DAO-ACK:	Destination Advertisement Object Acknoledgment
CSMA:	Carrier sense multiple access
DCM:	Módulo de Colección de Datos
DHM:	Módulo de Manejo de Datos
DVM:	Módulo de Visualización de Datos
DMM:	Módulo de Gestión de Base de Datos
APIM:	Módulo de API
DEF:	Marco de Exportación de Datos
JSON:	JavaScript Object Notation
API:	Application Programming Interface

Bibliografía

[1] Mitnick, K., & Simon, W. L. (2005). The art of intrusion : the real stories behind the exploits of hackers, intruders & deceivers / Kevin D. Mitnick, William L. Simon. (1st edition). Wiley Publishing, Inc.

[2] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, (2015). "A Review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT)", in 2015 Internet Technologies and Applications (ITA), pp. 219– 224, Sep, DOI: 10.1109/ITechA.2015.7317398.

[3] P. J. Ryan and R. B. Watson, (2017). "Research Challenges for the Internet of Things: What Role Can OR Play?," Systems, vol. 5, no. 1, pp. 1–34.

[4] M. Miraz, M. Ali, P. Excell, and R. Picking, (2018). "Internet of Nano-Things, Things and Everything: Future Growth Trends", Future Internet, vol. 10, no. 8, p. 68, DOI: 10.3390/fi10080068.

[5] E. Borgia, D. G. Gomes, B. Lagesse, R. Lea, and D. Puccinelli, (2016). "Special issue on" Internet of Things: Research challenges and Solutions".," Computer Communications, vol. 89, no. 90, pp. 1–4.

[6] K. K. Patel, S. M. Patel, et al., (2016). "Internet of things IOT: definition, characteristics, architecture, enabling technologies, application future challenges," International journal of engineering science and computing, vol. 6, no. 5, pp. 6122–6131.

[7] J. A. Chaudhry, K. Saleem, P. S. Haskell-Dowland, and M. H. Miraz, (2018). "A Survey of Distributed Certificate Authorities in MANETs," Annals of Emerging Technologies in Computing (AETiC), vol. 2, no. 3, pp. 11– 18, DOI: 10.33166/AETiC.2018.03.002.

[8] A. S. A. Daia, R. A. Ramadan, and M. B. Fayek, "Sensor Networks Attacks Classifications and Mitigation", Annals of Emerging Technologies in Computing (AETiC), vol. 2, no. 4, pp. 28–43, 2018, DOI: 10.33166/AETiC.2018.04.003.

[9] Wang, H., Zhang, D., and Shin, K. G. (2002). Detecting SYN flooding attacks. Proceedings of IEEE Infocom. IEEE Press, New York.

[10] Yaar, A., Perrig, A., and Song, D. (2004). SIFF: a stateless internet flow filter to mitigate DDoS flooding attacks. Proceedings of IEEE Symposium on Security and Privacy. IEEE Press, New York.

[11] Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P. (2005). Internet Denial of Service: Attack and Defense Mechanisms. Prentice Hall.

[12] D. Vernon, G. Metta and G. Sandini, (2007). "A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents", *IEEE Trans. Evol. Comput.*, vol. 11, pp. 151–180

[13] N. J. Nilsson, (2014). Principles of Artificial Intelligence, San Mateo, CA, USA:Morgan Kaufmann,

[14] R. P. Light, D. E. Polley and K. Börner, (2014). "Open data and open code for big science of science studies", *Scientometrics*, vol. 101, no. 2, pp. 1535–155.

[15] Dragan Perakovic, (2015). "Analysis of the IoT impact on volume of DDoS attacks". Semantic Scholar.

[16] Shivangi, Sangeeta, Reeta Rautela, "Predicting Depression with Social Media Images", (2024) IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC), pp.1-8.

[17] Niharika Panda, M. Supriya, (2022). "Blackhole Attack Impact Analysis on Low Power Lossy Networks", 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), pp.1-5,

[18] Ghada Aljufair, Mohammed Mahyoub, Abdulaziz S. Almazyad, (2023)."On Mitigating DIS Attacks in IoT Networks", 2023 18th Wireless On-Demand Network Systems and Services Conference (WONS), pp.104-109,

[19] Vikash Chaudhary, Sugandhi Midha, Rajesh Bahuguna, (2024). "Web Based Online Car Rental System", 2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC), pp.1-8.

[20] Prashant Maurya, Vandana Kushwaha, (2023). "Impact Analysis of Hello Flood Attack on RPL", Advanced Network Technologies and Intelligent Computing, vol.1798, pp.554

[21] Md Adil, Sugandhi Midha, V. K Srivastav, Kavita, (2024). "Detection and Prevention of DoS Attack in VANET Using Artificial Neural Network", 2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC), pp.1-7.

[22] Qian Jin, Yibo Qiao, Yining Chen, Cheng Zhuo, Qi Sun, (2024). "A SelectiveNet-Based Method for Defect Classification in Semiconductor Manufacturing", 2024 Conference of Science and Technology for Integrated Circuits (CSTIC), pp.1-3.

[23] Aji Gautama Putrada, Nur Alamsyah, Syafrial Fachri Pane, Mohamad Nurkamal Fauzan, Doan Perdana, (2023). "AUC Maximization for Flood Attack Detection on MQTT with Imbalanced Dataset", 2023 International Conference on Information Technology Research and Innovation (ICITRI), pp.133-138.

[24] Iuchi, K.; Matsunaga, T.; Toyoda, K.; Sasase, (2015) I. Secure parent node selection scheme in route construction to exclude attacking nodes from RPL network. In Proceedings of the 2015 21st Asia-Pacific Conference on Communications (APCC), Kyoto, Japan, 14–16; pp. 299–303.

[25] Tian, C.; Zhang, Y.; Zuo, W.; Lin, C.W.; Zhang, D.; Yuan, Y, (2022). A heterogeneous group CNN for image super-resolution. IEEE Trans. Neural Netw. Learn. Syst..

[26] Hamdi A. Al-Jamimi, (2024) "Synergistic Feature Engineering and Ensemble Learning for Early Chronic Disease Prediction", IEEE Access, vol.12, pp.62215-62233.

[27] Xiaofei Liu, Wuqiang Yang, Fan Meng, Tengchen Sun, (2024). "Material Recognition Using Robotic Hand With Capacitive Tactile Sensor Array and Machine Learning", IEEE Transactions on Instrumentation and Measurement, vol.73, pp.1-9.

[28] Vedha Krishna Yarasuri, Dhumsapuram Saikrishna Reddy, Puligundla Sai Muneesh, Ramabhotla Venkata Sai Kaushik, Thupalli Nanda Vardhan, K.L Nisha, (2022). "Developing Machine Learning Models for Cardiovascular Disease Prediction", 2022 2nd Asian Conference on Innovation in Technology (ASIANCON), pp.1-6.

[29] Saritha, Ritika Shetty, Manasa Devi, Akash Dhotre, Prema R Hanchinal, (2022)."Analysis on Air Quality and its Effects on Agriculture", 2022 3rd International Conference for Emerging Technology (INCET), pp.1-6.

[30] Rajasekar Mohan, Kartikey Mishra, Abhinav Reddy Bendrapu, Burra Venkata Vasishta, Boru Andreswar Reddy, (2022). "Prediction of Throughput of EXT WLANs through Machine Learning", 2022 International Conference for Advancement in Technology (ICONAT), pp.1-5.

[31] Patel, A., & Jinwala, D. (2022). A reputation-based RPL protocol to detect selective forwarding attack in Internet of Things. International Journal of Communication Systems, 35(1).

[32] Winters-Hilt, S. (2022). Informatics and machine learning :from Martingales to metaheuristics / Stephen Winters-Hilt. Wiley.

Webgrafía

[1] Centro Tecnológico de Seguridad (CETSE). (2024). Guía sobre Seguridad en Dispositivos IoT. [en línea]. Ministerio del interior. [Consultado: 3 de abril de 2024] <https://cetse.ses.mir.es/publico/cetse>

[2] Wikipedia, (2024). Multiplicadores de Lagrange [en línea]. Wikipedia. [Consulta: 8/4/24] <https://Multiplicadores_de_Lagrange>

[3] Wikipedia, (2024). Wolfe Duality [en línea]. Wikipedia. [Consulta: 8/4/24]. <https://Wolfe_duality>

[4] IBM, (2024). Medida de Impureza Gini [en línea]. IBM. [Consulta: 14/5/24]. <https://www.ibm.com>

[5] Recuero de los Santos, Paloma. (2021). Cómo interpretar la matriz de confusión: ejemplo práctico [en línea]. Telefónica Tech. [Consulta: 12/5/24]. <https://telefonicatech.com/blog/como-interpretar-la-matriz-de-confusion-ejemplo-practico>

[6] Statista, (2024). Dispositivos conectados (Internet de las cosas) a nivel mundial [en línea]. Statista. [Consulta: 10/04/2024].<https://es.statista.com/estadisticas/517654/prevision-de-la-evolucion-de-los-dispositivos-conectados-para-el-internet-de-las-cosas-en-el-mundo/>

[7] Omer Yoachimik. (2022). Ataque DDoS de Capa de Aplicación, distribución por país [en línea]. Cloudflare. [Consulta: 3/4/24] <https://blog.cloudflare.com/es-es/ddos-attack-trends-for-2022-q1-es-es>

[8] Wikipedia, (2022). Distancia euclidiana entre dos puntos [en línea]. Wikipedia [Consulta: 22/03/2024] <https://es.wikipedia.org/wiki/Distancia_euclidiana>

[9] Bealdung, (2023). Reducción dimensional en SOM [en línea]. Bealdung. [Consulta: 25/03/2024] <https://www.baeldung.com/cs/>

[10] Finne, Niclas, (2022) MSPSim version 0.9x [en línea]. Github. [Consulta: 25/03/2024] <https://github.com/contiki-ng/mspsim>.

Recursos

[1] Vmware, (2024). VMware Desktop Hypervisors [en línea]. Vmware by Broadcom. [Consulta: 16/4/24] <https://www.vmware.com/>

[2] Tsiftes, Nicolas, (2024). Contiki-NG contiki-ng [en línea]. Github. [Consulta: 16/4/2024] <https://github.com/contiki-ng/contiki-ng>

[3] Contiki-OS, (2024). An Introduction to Cooja [en línea]. Github. [Consulta: 16/4/24] <https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja>

[4] simonduq, (2017) UDP-Sink.c [en línea]. Github. [Consulta: 6/5/24] <https://github.com/contiki-os/contiki/blob/master/examples/ipv6/rpl-collect/udp-sink.c>

[5] simonduq, (2017) UDP-sender.c [en línea]. Github. [Consulta: 6/5/24] <https://github.com/contiki-os/contiki/blob/master/examples/ipv6/rpl-collect/udp-sender.c>

[6] g-oikonomou, (2020). Sensniff [en línea]. Github. [Consulta: 10/5/24] <https://github.com/g-oikonomou/sensniff>

[7] the-tcpdump-group, (2020). Biblioteca Libpcap [en línea]. Github. [Consulta: 14/5/24] <https://github.com/the-tcpdump-group/libpcap>

[8] alsawafi, (2017). IoTR-DS [en línea]. Github. [Consulta: 20/5/24] <https://github.com/alsawafi/IoT-DL-IDS>

[9] alsawafi, (2017). IoTR-DS DISAttack-5000s-updated2.csv [en línea]. Github. [Consulta: 24/5/24] <https://github.com/alsawafi/IoT-DL-IDS/blob/main/DISAttack-5000s-updated2.csv>

[10] Scikit-learn, (2024). sklearn.svm, sklearn.tree [en línea]. Scikit-learn. [Consulta: 2/6/24] <https://scikit-learn.org/stable/api/sklearn.preprocessing.html >

[11] Texas Instruments, (2024). Transceptor RF conforme a IEEE 802.15.4 de un solo chip de 2,4 GHz y ZigBee™ Ready [en línea]. Texas Instruments. [Consulta:1/6/24] <https://www.ti.com/product/es-mx/CC2420#description>

[12] IBM, (2013). ¿Qué es JDBC? [en línea]. IBM. [Consulta: 19/5/24] < https://www.ibm.com/docs/es/informix-servers/12.10?topic=started-what-is-jdbc>

[13] Scikit-learn, (2024). Support Vector Machines [en línea]. Scikit-learn. [Consulta: 2/6/24] <https://scikit-learn.org/stable/modules/svm.html>

[14] Scikit-learn, (2024). Ensembles: Gradient boosting, random forests, bagging, voting, stacking [en línea]. Scikit-learn. [Consulta: 2/6/24] <https://scikit-learn.org/stable/modules/ensemble.html>

[15] Datacamp, (2024). Tutorial sobre máquinas de vectores de soporte con Scikit-learn [en línea]. Datacamp. [Consulta: 5/6/24]< https://www.datacamp.com/es/tutorial/svm-classification-scikit-learn-python>

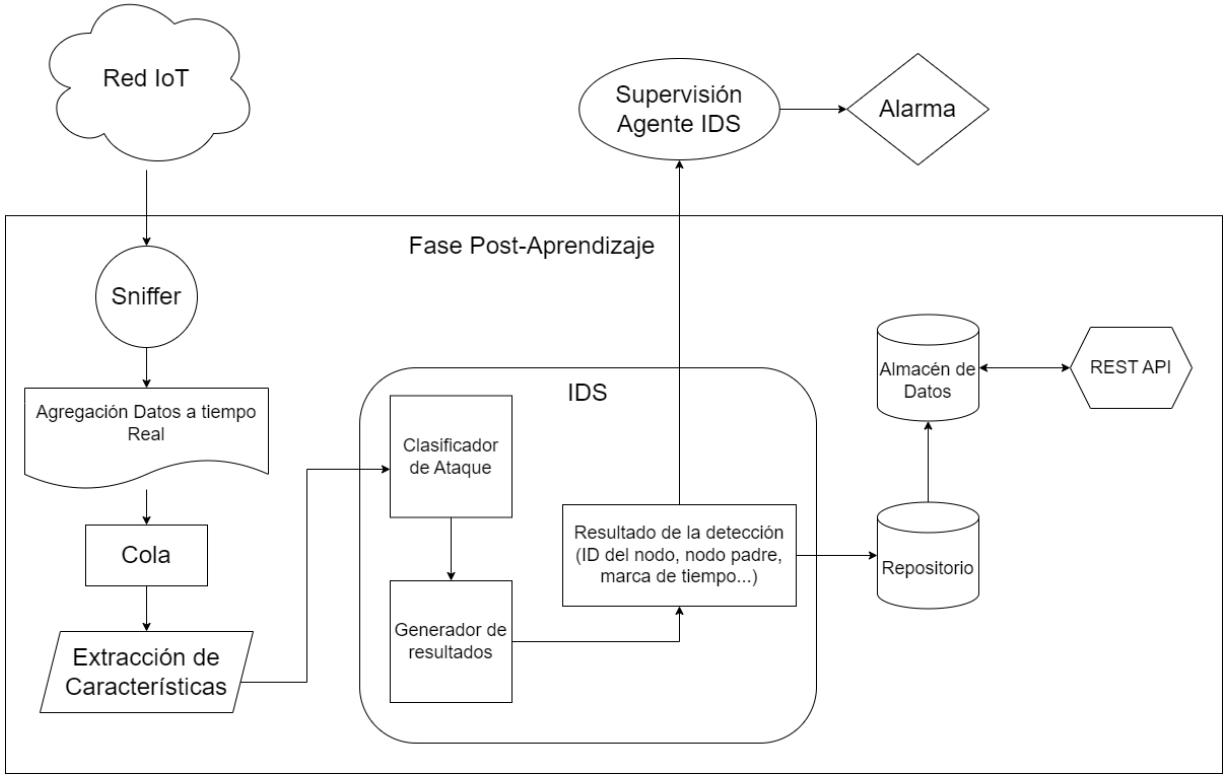
[16] Datacamp, (2024). Random Forest Classification with Scikit-Learn [en línea]. Datacamp. [Consulta: 5/6/24]<https://www.datacamp.com/tutorial/random-forests-classifier-python>

[17] alsawafi, (2017). IoTR-DS-RPLNormal-updated1.csv [en línea]. Github. [Consulta: 24/5/24] <https://github.com/alsawafi/IoT-DL-IDS/blob/main/RPLNormal-updated1.csv>

[18] CodeLite, (2024). What is CodeLite? [en línea]. CodeLite. [Consulta: 1/5/24] <https://codelite.org/

Anexo

Esquema de flujo general de la Fase de Post-Aprendizaje.



Flujo General