UNIVERSIDAD DE CANTABRIA

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Sistemas Embebidos en Red Seguros

Memoria presentada para optar al título de
DOCTOR EN CIENCIAS, TECNOLOGÍA Y COMPUTACIÓN
POR LA UNIVERSIDAD DE CANTABRIA

por Álvaro Díaz Suárez,
Ingeniero en Informática

Director: Pablo Pedro Sánchez Espeso

Santander, Abril 2017

Agradecimientos

Deseo expresar mi más sincero agradecimiento a mi director de tesis Pablo Sánchez, por el apoyo, dedicación, confianza y amistad que me ha demostrado desde que empecé a trabajar con él. Sin su ayuda hubiera sido imposible realizar esta tesis.

También quiero agradecer a Eugenio Villar la visión de mercado que siempre aporta a los trabajos, así como la confianza que depositó en mí permitiéndome entrar a trabajar en el grupo cuando apenas había terminado mis estudios de ingeniería informática.

Por supuesto, quiero y debo agradecer a todos los compañeros que han pasado por el grupo y han colaborado en el desarrollo de distintas partes de esta tesis. En especial a Héctor Posadas y Luis Díaz, por el esfuerzo de desarrollar y mantener SCoPE y VIPPE, así como toda la ayuda que me han aportado durante estos años. También debo agradecer la ayuda recibida de los compañeros que ya abandonaron el grupo, especialmente de: Javier Barreda, Javier González, Raúl Diego, Pablo Peñil, Alejandro Nicolás, Fernando Herrera y Pablo González. Todos ellos han aportado su granito de arena a esta tesis. Quiero también reconocer el trato y el trabajo del personal de administración (Sara y Bea) del departamento TEISA.

Y, por supuesto, a mis padres Antonio y Milagros, y a mi familia. Porque sin ellos nunca hubiese llegado a ser lo que soy.

A Sara, por todo…

Tabla de contenidos

Lista de Acrónimos

API	*Application Programming Interface* -Interfaz de programación de aplicaciones
AES	*Advanced Encryption Standard* – Estándar de encriptación avanzada
AES-128	Estándar de encriptación avanzada con clave de 128 bits
AES-192	Estándar de encriptación avanzada con clave de 192 bits
AES-256	Estándar de encriptación avanzada con clave de 256 bits
App	Aplicación
ASCII	*American Standard Code for Information Interchange* – Código Estándar Estadounidense para el Intercambio de Información
BBN	*Broad Band Noise* -Banda-ancha
CAGR	*Compound annual growth rate* – Tasa anual compuesta de crecimiento
CPU	*Central Processing Unit* – Unidad Central de procesamiento
CTS	*Clear to Send*
dB	decibelio
dBm	decibelio-milivatio
DES	*Data Encryption Algorithm*
DFT	*Discrete Fourier Transform* – Transformada discreta de Fourier
DoS	*Denial of service*
ECC	*Elliptic curve cryptography* – Criptografía de Curva Elíptica
EEPROM	*Electrically Erasable Programmable Read-Only Memory* - ROM programable y borrable eléctricamente
ENISA	*European Union Agency for Network and Information Security*
E-R	*Entity relationship* – Entidad-Relación
FUT	*Function Under Test* – Función para testear

GB	Gigabyte
GHz	Gigahercio
GSM	*Global System for Mobile Communications*
GPRS	*General Packet Radio Service*
HASH	Función resumen
HEX	Hexadecimal
HMAC	*Hash Message Authentication Code* – Código de autentificación de mensajes en clave hash
HW	*Hardware*
Hz	*Herzios*
IoT	*Internet of Things* – Internet de las cosas
ISS	*Instruction Set Simulator*
J	Julio
JPEG	Estándar de compresión y codificación de archivos e imágenes fijas
JPG	Estándar de compresión y codificación de archivos e imágenes fijas
JRC	*Journal Citation Reports*
LCD	*Liquid Cristal Display*
mA	Miliamperios
MAC	*Message Authentication Code – Código de autentificación de mensaje*
MD5	*Message Digest Algorithm 5* – Algoritmo de Resumen del Mensaje 5
MMU	*Memory Management Unit* – Unidad de gestión de memoria
mW	Milivatio
mWh	Milivatio-hora
NIST	*National Institute of Standards and Technology* – Instituto Nacional de Estándares y Tecnología
OR	Disyunción lógica

OS	*Operating System* – Sistema operativo
OTAP	*Over The Air Programming* – Programación por el aire
PBN	*Partial Band Noise* – Banda-parcial
PC	*Personal Computer* – Ordenador
PGM	*Portable Graymap Format*
PIR	*Passive infrared sensor* – Sensor infrarrojo pasivo
RAM	*Random Access Memory* - Memoria de acceso aleatorio
RC4	*Rivest Cipher* 4
RF	Radio Frecuencia
ROM	*Read-Only Memory* – Memoria de solo lectura
RSA	*Rivest, Shamir y Adleman* – Sistema criptográfico de clave pública
RTOS	*Real Time Operating System* – Sistema operativo de tiempo real-
RTS	*Require To Send*
SD	*Secure Digital*
SDK	*Software Development Kit*
SEM	*Security Estimation Metric* – Métrica de estimación de la seguridad
SHA-1	*Secure HASH Algorithm 1*
SOA	*State-Of the Art* – Estado del arte
SRAM	*Static Random Access Memory* – Memoria estática de acceso aleatorio
SW	Software
TDES	Trip*le Data Encryption Algorithm* – Algoritmo de triple cifrado del DES
TTM	*Time To Market* - Tiempo hasta el mercado
TXT	Archivo de texto simple
UML	*Unified Modeling Language* – Lenguaje unificado de modelado
UML-RT	*Unified Modeling Language* - *Real Time* – Lenguaje unificado de modelado de tiempo real

uP	Microprocesador
UVM	*Universal verification methodology* – Metodología de verificación universal
VIPPE	*Virtual Parallel platform for Performance Estimation*
WLAN	*Wireless Local Area Network* – Red de área local inalámbrica
WSN	*Wireless Sensor Network* -Red de sensores inalámbrica
XML	*eXtensible Markup Language* – Lenguaje de Marcas Extensible
XOR	Or exclusiva

Lista de Figuras

Lista de Tablas

1. Motivación y objetivos

La seguridad de los sistemas electrónicos embebidos es un aspecto que preocupa cada vez más, tanto a los usuarios como a los gobiernos y fabricantes. El creciente uso de estos dispositivos y la expansión de "Internet de las cosas" (IoT, *Internet of Things*) ha hecho que la importancia de la seguridad de los sistemas embebidos haya crecido exponencialmente durante los últimos años, de tal forma que el constante aumento del número de usuarios y dispositivos ha traído consigo un fuerte incremento del número, tipología y gravedad de los ataques que violan la seguridad de estos sistemas.

Como constatan el Centro Criptológico Nacional (en sus informes de ciberamenazas y tendencias [1] [2] [3]) y ENISA (*European Union Agency for Network and Information Security*) [4], el número de ataques y su sofisticación aumentan año tras año (Figura 1). Por lo tanto, la generalización del uso de los sistemas embebidos y aplicaciones de IoT, incrementan las posibilidades de ataque y los beneficios potenciales derivados de los mismos, lo que constituye uno de los mayores estímulos para los atacantes [1] y un gran reto para los diseñadores, obligados a considerar la seguridad como un aspecto esencial del desarrollo del producto.

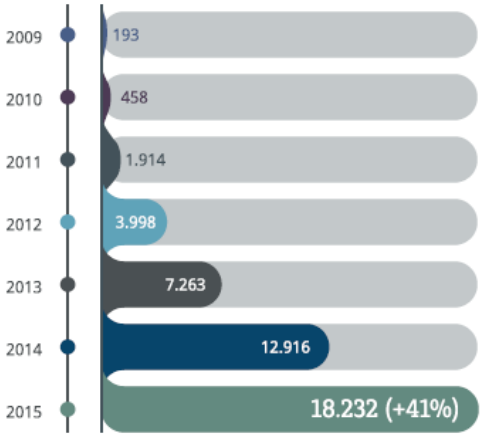


Figura 1: Evolución de incidentes gestionados por el CCN-CERT

Las redes de sensores inalámbricas son uno de los sistemas embebidos más vulnerables. Una Red de Sensores Inalámbrica (WSN: Wireless Sensor Network) es un conjunto de elementos autónomos (nodos o "*motes*") interconectados de modo inalámbrico, que colaboran con el objetivo de resolver una tarea común. El término "*mote*" (en castellano "mota de polvo") lo empezó a utilizar la Universidad de Berkeley [5] hacia el año 2000, y permite reflejar en una única palabra las dos características principales de este tipo de dispositivos: su pequeño tamaño y que pueden estar situados en cualquier lugar (como una mota de polvo). Estas redes están compuestas por pequeños dispositivos electrónicos que capturan información del entorno mediante sensores y la retransmiten, a través de la red, hasta un punto de control, que registra los valores observados para tomar decisiones. La red está formada por nodos sensores o "*motes*", que se conectan entre ellos de manera inalámbrica, y nodos pasarela (o *gateways*) que permiten comunicar la red con otras redes (por ejemplo Internet) o sistemas externos. Normalmente este tipo de redes incluyen un gran número de nodos (de cientos a miles) de bajo coste, bajo consumo, elevados tiempos de operación sin mantenimiento ni cambio de batería, con recursos limitados y con múltiples sensores que comúnmente operan en entornos hostiles de forma autónoma y desatendida [6].

Durante los últimos años ha aumentado enormemente el despliegue de WSN. De acuerdo con el análisis del sector realizado por IDTechEx [7], el mercado de redes de sensores inalámbricas crecerá desde los $0.45 billones de dólares en 2012, hasta los $2 billones en 2022. Según el informe [8], se espera que dicho

mercado crezca alrededor de un 14% anual hasta 2022. La Figura 2 muestra una estimación del CAGR (*Compound Annual Growth Rate,* Tasa anual compuesta de crecimiento) para este mercado [8]. Análisis de diferentes organizaciones ([9] [10] y [11]) proporcionan estimaciones con datos similares, que coinciden en la tendencia alcista a largo plazo del mercado de redes de sensores inalámbricas. Estos análisis muestran la importancia que estas redes están adquiriendo y permiten hacerse una idea del gran impacto que las mismas tienen y van a tener en nuestra vida diaria.

Actualmente, las redes de sensores inalámbricas están proliferando en mercados tan dispares como el de la minería, alimentación, aplicaciones militares, industria farmacéutica, salud, automoción, gas, petróleo, entretenimiento o transporte entre otros. En estas aplicaciones, estas redes permiten medir y controlar infinidad de parámetros (como pueden ser la temperatura, presión, humedad o flujo de gases y líquidos) al tiempo que, por ejemplo, pueden capturar imágenes, transmitir información a los usuarios o detectar situaciones anómalas.

Figura 2: Previsión del mercado de redes de sensores inalámbricas (2016-2022)

El crecimiento y la expansión de este tipo de redes inevitablemente amplían las posibilidades de los ciberdelincuentes de violar su seguridad. La mayoría de redes de sensores inalámbricas suelen estar desplegadas en zonas desprotegidas y muy vulnerables, lo que unido a sus especiales características (bajo consumo, baja capacidad de cómputo, etc.) aumenta enormemente las posibilidades de ser atacadas.

Aunque algunos de los riesgos de seguridad conocidos en redes tradicionales (como Internet) son aplicables a las redes de sensores inalámbricas, estas últimas

tienen características específicas que las hacen estar especialmente desprotegidas frente a los ataques. Normalmente las redes de sensores son desplegadas en entornos desatendidos y sin vigilancia, con un canal de comunicación inalámbrico, compartido y con poca protección. Si a esto se añade la reducida capacidad de cómputo del nodo, su estricta gestión de la energía y la propia complejidad de la red, el resultado es un sistema especialmente vulnerable. Además los atacantes normalmente tienen acceso directo al hardware de cada nodo, lo que les proporciona un gran número de posibles vías para atacar los dispositivos físicos o la red. Por todo ello, es necesario introducir cada vez más requisitos de seguridad en la especificación de redes de sensores, al tiempo que mejoran los mecanismos utilizados para evitar ataques. Los nuevos requisitos dependerán del nivel de seguridad que la red necesite, ya que no es lo mismo que la red maneje información relativa a la salud de personas a que gestione los aparcamientos disponibles en una *SmartCity*. Ambos sistemas deberán ser protegidos pero en diferente medida y con diferente coste, ya que la peligrosidad o impacto de un incidente sería distinto en cada red.

Uno de los retos más importantes a los que se enfrentan los desarrolladores de WSN es la gestión de la infinidad de parámetros que definen cada red: número de nodos, tipo de despliegue, topología de la red, aplicación software que ejecutan los nodos, posibles riesgos de seguridad, vulnerabilidad del entorno, esperanza de vida autónoma de los nodos, etc. A la luz de las especiales características de estos sistemas es necesario proveer a los desarrolladores de un conjunto de métodos y herramientas que faciliten el diseño y análisis de redes de sensores inalámbricas, de manera que cuando llegue la hora de desplegar la red, esta esté en condiciones de trabajar inmediatamente, sin errores de diseño y con mecanismos de protección adecuados para evitar ataques. Gracias a estos métodos lograremos un diseño más óptimo, que permitirá mejorar aspectos esenciales de las redes de sensores, como son:

- **Reducir el tiempo de desarrollo de la red y el coste de mantenimiento.** Uno de los mayores problemas que tienen actualmente los desarrolladores de WSN es la falta de un sistema que les permita saber si la red funciona correctamente antes de su despliegue. Por ello, los problemas de diseño

son detectados una vez que la red está en funcionamiento, aumentando el coste del desarrollo. El disponer de métodos de análisis del comportamiento de la red durante el proceso de diseño permite no solo reducir el coste de desarrollo, sino también el coste de mantenimiento de la red una vez desplegada.

- **Mejorar la seguridad de la WSN**. Al ser las redes de sensores sistemas autónomos que trabajan en entornos hostiles, es necesario desarrollar mecanismos que permitan reducir posibles incidencias de seguridad y proteger la red contra posibles ataques. Estos mecanismos pueden dotar al sistema de la capacidad de detectar ataques una vez realizado el despliegue. Hay que tener en cuenta que las violaciones de seguridad normalmente implican un importante coste de mantenimiento del sistema, que debe ser gastado cada vez que haya que reparar un defecto de seguridad o tratar una incidencia.

- **Aumentar la vida de la red**. El consumo eficiente de energía es una de las prioridades de una WSN, ya que en la mayoría de las aplicaciones los nodos son alimentados por baterías. Normalmente la recarga o la sustitución de la batería es impracticable o muy costosa, por lo que la optimización del consumo de energía prolonga el funcionamiento de los nodos y alarga la vida de la red. El análisis del comportamiento del sistema antes del despliegue permite ajustar el comportamiento de los nodos para maximizar su ciclo de vida, reduciendo al máximo el consumo. Además, la protección frente a ataques ayuda a reducir e incluso evitar uno de los efectos más dañinos de los fallos de seguridad en este tipo de redes: el aumento del gasto de energía con el consiguiente acortamiento de vida del nodo y de la red.

Para mejorar la seguridad de las redes de sensores inalámbricas, esta tesis propone distintas técnicas y herramientas que pueden ser utilizadas durante todas las fases del proceso de diseño. Además, se han desarrollado novedosos mecanismos de seguridad que ayudaran a proteger y aumentar la vida de la red. Una de las premisas con las que se han desarrollado estas técnicas y herramientas

es que las mismas sean fácilmente integrables en las diferentes fases del proceso de diseño.

El proceso de diseño se puede definir como un conjunto de tareas de desarrollo, análisis y verificación que, combinadas adecuadamente, permiten producir un sistema funcional. Existen múltiples propuestas de procesos de diseño en la literatura, con distintas tareas y orden de aplicación. Sin embargo, la mayoría de los procesos tienen unas etapas comunes con unas tareas bien definidas. Por ejemplo la Figura 3 presenta el típico modelo del proceso de diseño en cascada, el cual está compuesto por 5 etapas bien diferenciadas:

1. **Especificación del sistema:** En esta fase se definen los servicios y funcionalidades que tendrá que ofrecer el sistema a diseñar, así como los requisitos que deberá cumplir. También se define el entorno del sistema y las características básicas de su relación con el mismo.

2. **Fase de diseño:** Se descompone y organiza el sistema en módulos, definiéndose la funcionalidad de cada uno y su relación con el resto. Dichos módulos pueden desarrollarse por separado, aprovechando las ventajas del desarrollo en equipo.

3. **Fase de implementación:** En esta etapa se desarrolla el código (en el caso de un desarrollo software) específico de cada módulo, teniendo en cuenta la especificación y los requisitos del sistema

4. **Fase de verificación:** Esta fase está destinada a comprobar que el sistema cumple con los requisitos especificados. Para ello se comprueba que el código está libre de errores, entendiendo como tales aquellos comportamientos que violan la especificación o funcionalidad esperada del sistema.

5. **Mantenimiento y soporte al producto:** Esta etapa incluye las actividades de desarrollo que se realizan desde que se entrega el sistema al usuario (o se despliega la red, en el caso de WSN) hasta que el sistema deja de

operar. En esta fase se solucionan problemas tales como que el producto no cumpla con todas las expectativas del usuario, surjan errores durante su operación en campo o se necesite modificar su funcionalidad.

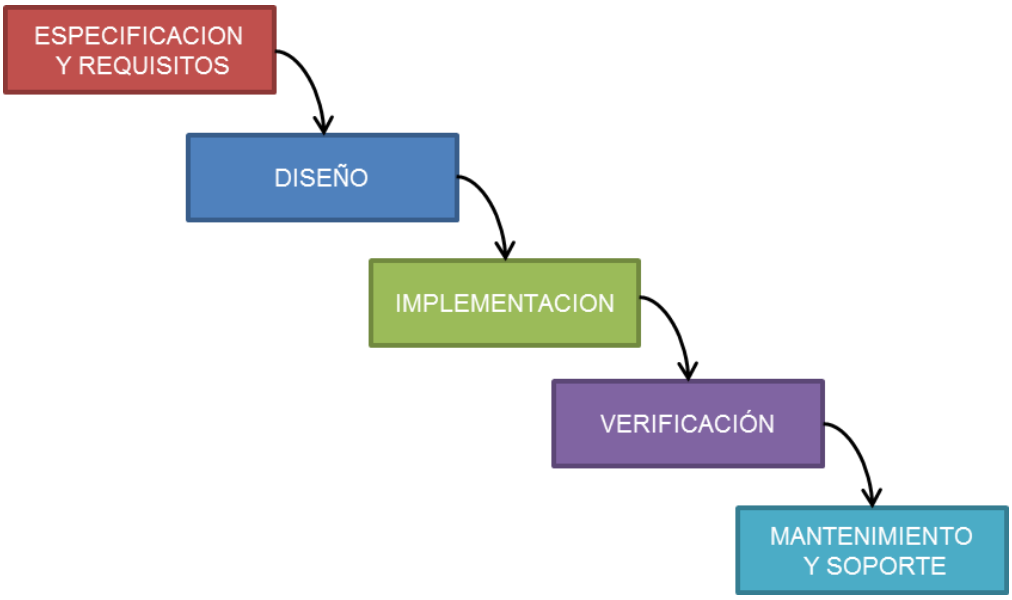


Figura 3: Método tradicional de cascada

El modelo en cascada es uno de los esquemas clásicos de desarrollo aunque en la literatura existen múltiples variantes como el modelo iterativo, la metodología ágil o el modelo en V [12].

1.1. Objetivo de la tesis

Este trabajo nace de la necesidad de aumentar la seguridad y reducir el coste de desarrollo de sistemas embebidos en red (*Networked Embedded Systems*), como es el caso de las redes de sensores inalámbricas que se utilizan en aplicaciones de "Internet de las cosas" (*Internet of Things*). Una premisa importante es que las técnicas propuestas para mejorar la seguridad no deben aumentar significativamente el consumo de energía para no reducir el tiempo de vida de las baterías que normalmente alimentan a este tipo de sistemas.

Por lo tanto, el objetivo principal de esta tesis es desarrollar técnicas que permitan reducir el coste de diseño de redes de sensores inalámbricas y aumenten su nivel de seguridad, reduciendo su impacto en el consumo de energía. Este objetivo global se puede descomponer en cinco sub-objetivos:

1. Desarrollar técnicas de estimación del funcionamiento del sistema durante el proceso de diseño. Para tal fin se desarrollará un simulador que permitirá estimar el funcionamiento de la red teniendo en cuenta el hardware y el software de cada nodo. Para evaluar la seguridad, en el simulador se podrán modelar ataques, lo que permitirá conocer sus efectos y evaluar su peligrosidad. El simulador, además de evaluar la funcionalidad, podrá estimar aspectos no-funcionales como el consumo de energía y el tiempo de ejecución/respuesta.

2. Proponer una metodología de diseño del software del nodo (firmware) que minimice el impacto de los ataques a la red. Esta metodología hará uso del simulador con modelado de atacantes, de forma que los desarrolladores podrán proponer mecanismos de detección de ataques y diseñar contramedidas a partir de las estimaciones del comportamiento de la red atacada.

3. Establecer una métrica que permita estimar la seguridad de una red. Esta métrica podrá integrarse en el entorno de análisis de prestaciones, de forma que su seguridad puede ser evaluada conjuntamente con otros parámetros no-funcionales, como el consumo de energía o el tiempo de ejecución.

4. Definir e implementar mecanismos que mejoren la seguridad del sistema, con un bajo impacto en su consumo. Estos mecanismos se focalizarán en puntos débiles de las redes inalámbricas, como son el arranque del nodo (*booting*) y su proceso de actualización en campo (OTAP, *Over The Air Programming*).

5. Reducir el tiempo del desarrollo del sistema mediante la elaboración de una metodología que permita verificar su funcionalidad durante todas las fases

del proceso de diseño. El objetivo primordial de esta tarea es proporcionar un método que permita realizar test funcionales y no-funcionales que puedan ser reutilizados en todas las etapas del proceso de diseño. Estos test podrán ser ejecutados incluso cuando la plataforma hardware final no esté disponible, utilizando para ello plataformas virtuales.

1.2. Estructura de la tesis

Esta tesis propone diversas técnicas para reducir el tiempo de desarrollo y mejorar la seguridad de sistemas embebidos en red. Un esquema de los mecanismos desarrollados se muestra en la Figura 4. Dicha figura también indica la sección en donde se presenta cada técnica y será utilizada como referencia en la descripción de la estructura de la tesis que se realiza a continuación. La tesis se ha divido en 3 partes:

1. La primera parte describirá un entorno de análisis de prestaciones para redes de sensores inalámbricas (cuadro morado en la Figura 4). Mediante el uso del simulador propuesto se podrá estimar el comportamiento de la red antes de su despliegue, así como diversos parámetros no-funcionales (consumo y tiempo de ejecución). El simulador tendrá en cuenta el hardware del nodo, así como el software que se ejecuta en cada procesador. También tendrá en cuenta la topología de la red y las características del canal inalámbrico de comunicaciones, entre otros parámetros. La descripción del simulador se realizará en el capítulo 2 de la tesis. Para poder evaluar la seguridad en la red, el simulador integrará modelos de ataques de forma que sea posible evaluar los efectos de estos sobre la red diseñada. La descripción del simulador con modelos de ataques será realizada en el capítulo 3 de la tesis. Además de evaluar la funcionalidad del sistema, el simulador podrá estimar la seguridad de la red mediante una nueva métrica desarrollada en esta tesis (métrica “SEM”). Dicha métrica será presentada en el capítulo 4 de la tesis.

2. La segunda parte de la tesis presenta técnicas que permiten el desarrollo de firmware seguro y eficiente para redes de sensores inalámbricas (cuadro naranja en la Figura 4). La primera metodología utiliza el análisis del impacto de los ataques (descrito en el capítulo 3) para diseñar *firmware* (software que se ejecuta en el nodo) que soporte el impacto de los ataques (*attack-aware software*). Con objeto de facilitar la lectura de la tesis, esta metodología ha sido descrita en la sección 3.8 de la tesis, dentro del capítulo dedicado a la simulación de ataques (capítulo 3). En el capítulo 5, se describen otras dos técnicas que mejoran la seguridad y que normalmente se integran entre los servicios que ofrece el RTOS (*Real-Time Operating System*) del nodo. La primera técnica es un método que permite la actualización parcial del firmware una vez desplegada la red (OTAP, *Over-The-Air-Programming*). Esta técnica permite reducir drásticamente el tiempo de actualización del firmware de los nodos de la red, y con ello el consumo de energía. Además, este método posibilita tener control de versiones dentro del nodo, permitiendo modificar la versión de software sin apenas incrementar el consumo. El segundo método permite un arranque seguro del nodo (*secure boot*). Esta característica (como señala [1]) es esencial en cualquier sistema seguro, ya que el arranque suele ser uno de los puntos más vulnerables a ataques. A la necesidad de un arranque seguro se añade la limitación del consumo que caracteriza a las redes de sensores. Por esta razón, en esta tesis se propone un mecanismo de arranque que permite balancear entre seguridad y consumo. Esta técnica será presentada en la sección 5.2.

3. La tercera parte de la tesis presenta una metodología de verificación de sistemas embebidos que permite reutilizar los test durante las distintas fases del proceso de diseño (cuadro rojo en la Figura 4). El objetivo de la metodología es poder realizar la verificación del sistema de manera concurrente con las etapas de diseño e implementación, permitiendo de esta forma detectar y corregir errores desde las primeras fases del proceso de diseño. La metodología propuesta no se limitará a comprobar aspectos funcionales del sistema, sino que tendrá en cuenta la arquitectura de la

plataforma y validará aspectos no-funcionales. En ocasiones no es posible verificar el software del sistema por estar diseñando de forma concurrente la plataforma hardware. La metodología propuesta evita esta limitación mediante el uso de una plataforma virtual, que permite estimar las prestaciones y evaluar la funcionalidad del sistema antes de disponer del hardware físico. La metodología de verificación será presentada en el capítulo 6 de la tesis.

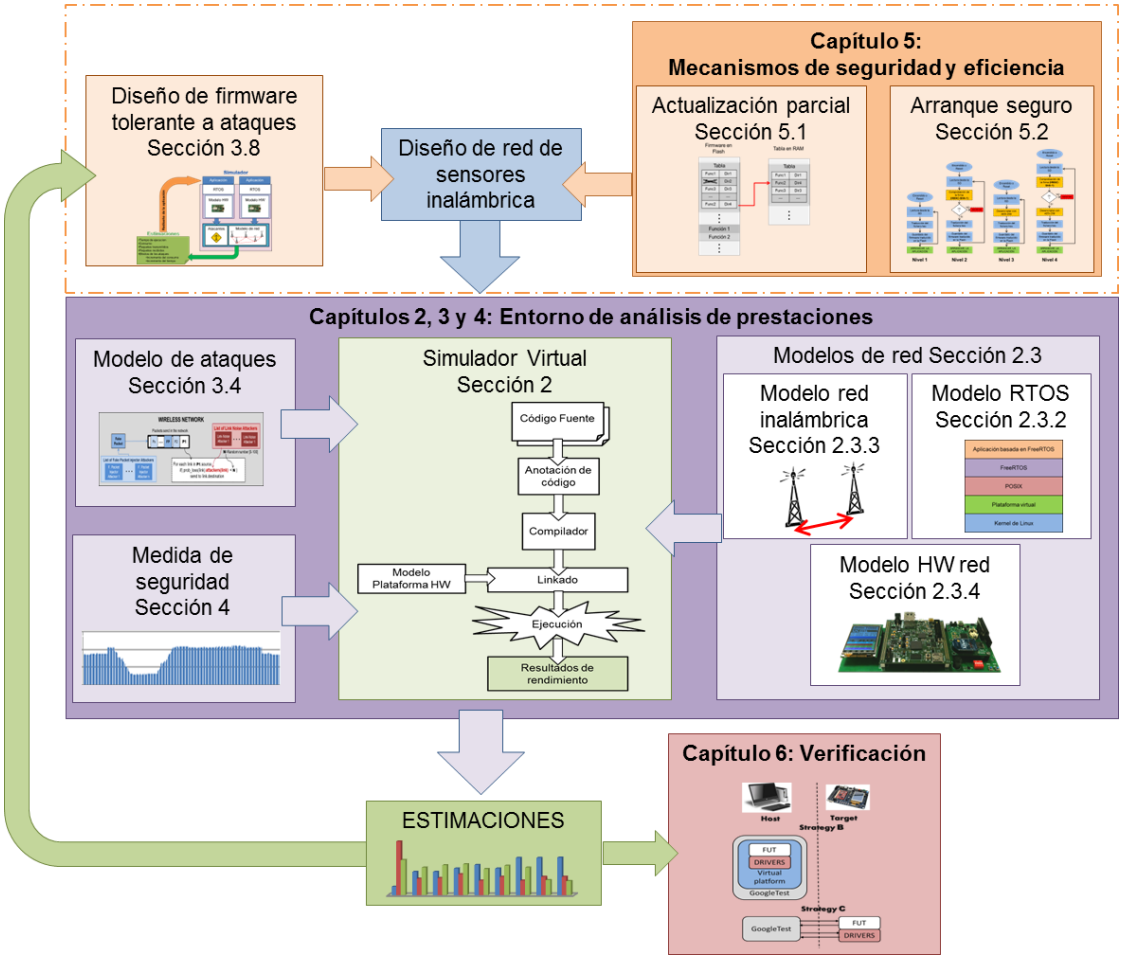


Figura 4: Resumen de aportaciones principales de la tesis

Para finalizar, el capítulo 7 presenta las conclusiones de este trabajo así como las publicaciones a las que él mismo ha dado lugar. También se indica en dicho capítulo los proyectos de investigación a nivel europeo y nacional en los cuales se ha hecho uso de resultados obtenidos en el marco de la presente tesis.

2. Simulación de redes de sensores inalámbricas

2.1. Introducción

A la hora de desarrollar una red de sensores inalámbrica, el diseñador no solo se enfrenta a las dificultades que plantean los estrictos requisitos del sistema (como por ejemplo tener que operar de forma autónoma durante años alimentado por batería) o a la propia complejidad funcional del mismo, sino también a las condiciones que impone el propio mercado como son el bajo coste por nodo y reducido TTM (*Time To Market*, tiempo hasta el mercado). Para reducir el tiempo de diseño y de llegada al mercado (TTM) se precisan simulaciones rápidas y precisas del comportamiento de la red desde las primeras fases del proceso de diseño. De esta forma es posible identificar y corregir los errores en cuanto se producen, sin tener que esperar hasta que la red este desplegada y funcionando.

El objetivo de esta sección es presentar el simulador desarrollado para evaluar el comportamiento de una red de sensores inalámbrica. Esta herramienta de simulación está basada en una técnica de co-simulación Hardware/Software que permite realizar estimaciones a nivel sistema durante las primeras fases del desarrollo. El simulador proporciona numerosas ventajas durante el proceso de diseño, entre las que destacan:

- Poder verificar los programas y las plataformas hardware desde las primeras etapas del desarrollo. La posibilidad de simular de manera rápida y precisa el comportamiento del sistema proporciona información que permite evaluar diferentes alternativas de diseño (como la arquitectura de red) o sirve de guía para modificar el software o hardware de cada nodo, permitiendo buscar una mejor relación entre el rendimiento y el coste del sistema.

- Poder incrementar la duración de la batería y, por ello, el ciclo de vida de la red. En este tipo de sistemas, el apagado de un nodo por falta de energía puede causar el aislamiento de una parte de la red. Por lo tanto, estimar el consumo de cada nodo provee de información crucial para diseñar una red con un ciclo de vida acorde con las necesidades del cliente.

- Poder simular la red con el tráfico realmente generado por la aplicación software de los nodos. Una gran parte de los simuladores de redes de sensores existentes utilizan un tráfico de red estadístico, por lo que es muy complicado obtener medidas reales del comportamiento del sistema. El simulador presentado permite estimar el tráfico de red real que produce el software que se ejecuta en los nodos, lo que mejora la estimación.

- Poder evaluar el impacto del sistema operativo que gestiona los nodos de la red. Los nodos de la red de sensores normalmente utilizan un RTOS (*Real-Time Operating System*) para gestionar las tareas software que precisa la aplicación. El simulador desarrollado permite estimar el efecto del RTOS en la ejecución del software (*firmware*) del nodo.

2.2. Simulación de redes de sensores: Estado del arte

Como se ha comentado anteriormente, la simulación de redes de sensores desde las primeras etapas del proceso de diseño permite verificar la funcionalidad del sistema y guiar la modificación del software y/o hardware de los nodos durante el desarrollo del producto. Sin embargo, la mayoría de los entornos de simulación de redes tradicionales no capturan la funcionalidad del sistema en detalle, ya que no tienen en cuenta el hardware y/o el software de cada nodo. Si esta información no se tiene en cuenta o es ignorada por la simulación, no es posible estimar con precisión aspectos como el consumo de energía o los tiempos de ejecución. Trabajos como [13], [14], [15], [16], [17], [18] y [19] reflejan el estado del arte en simulación de redes de sensores inalámbricas. En la bibliografía también aparecen numerosos ejemplos de simuladores de redes como Cooja [20], Castalia [21], NS-2 [22], NS-3 [23] OMNET++ [24], GloMoSim [25], TOSSIM [26] y Avrora [27]. NS-2, NS-3 y OMNET++ son simuladores de redes basados en eventos discretos. NS-2 se utiliza activamente en la investigación de redes móviles e implementa una amplia gama de protocolos, tanto de redes cableadas como de redes inalámbricas. NS-3 es la nueva versión de NS-2, diseñada para soportar todo el flujo de trabajo de la simulación, desde la configuración de la red hasta la recolección y análisis de tramas. OMNET++ es un simulador modular de eventos discretos de redes orientado a objetos, usado habitualmente para modelar el tráfico de redes de telecomunicaciones. Igualmente permite simular protocolos, sistemas multiprocesadores y distribuidos, evaluar arquitecturas hardware, estimar el rendimiento de sistemas software y, en general, simular cualquier sistema que pueda modelarse mediante eventos discretos. Normalmente OMNET++ suele ser usado para verificar algoritmos distribuidos y protocolos sobre canales de radio realistas. GloMoSim (*Global Mobile Information System Simulator*) es una librería de simulación escalable para redes de sistemas inalámbricas construida sobre el entorno de simulación PARSEC [28]. Otra opción para verificar una WSN es TOSSIM, un simulador de eventos discretos a nivel de bit para redes de sensores basadas en el sistema operativo TinyOS [29]. TOSSIM captura el entorno y las

interacciones de la red con una granularidad a nivel de bit en vez de a nivel de paquete. TOSSIM se ha diseñado específicamente para aplicaciones basadas en el sistema operativo TinyOS y que son ejecutadas en una plataforma específica. Otra herramienta de simulación es Avrora, la cual permite simular ejecuciones de programas en microcontroladores con precisión de ciclo de reloj. Sin embargo, Avrora solo permite simular dos tipos muy específicos de plataforma HW. Existen extensiones de los simuladores anteriormente comentados, como por ejemplo [30], donde se presenta una extensión de OMNET++ que provee una interfaz de desarrollo mediante el uso de modelos UML-RT.

En [31], los autores presentan una técnica para modelar y analizar el rendimiento del SW que se ejecuta en los nodos de una red de sensores. Dichos nodos están programados con NesC, un lenguaje orientado a componentes y que está especialmente diseñado para desarrollar aplicaciones de redes de sensores con sistema operativo TinyOS. J-Sim [32] es un entorno de simulación basado en componentes desarrollado en Java. La principal limitación de este simulador es su baja eficiencia. Otro simulador (UWSim) es presentado en [33]. UWSim es específico para redes de sensores submarinas. El simulador Shawn [34] es un framework de simulación cuya idea central es modelar los efectos a bajo nivel de una red de sensores inalámbrica mediante modelos abstractos e intercambiables, de manera que se puedan utilizar en la simulación de grandes redes en un tiempo razonable. Otros simuladores, como Prowler [35] y JProwler [36] utilizan modelos probabilísticos. Prowler está escrito en Matlab mientras que JProwler está escrito en Java. Otra opción es ATEMU [37], el cual centra sus simulaciones en dispositivos hardware específicos (Crossbow AVR/Mica2).

La mayoría los simuladores de red descritos anteriormente no capturan la funcionalidad de los nodos con detalle ya que la simulación no considera el HW y/o el SW de cada nodo. Sin esta información es imposible estimar con precisión el comportamiento y los aspectos no-funcionales (por ejemplo consumo) del sistema.

Para poder evaluar con precisión el comportamiento del sistema, es necesario poder simular el nodo completo, mediante una co-simulación de la parte HW y SW. Uno de los métodos de co-simulación más utilizados está basado en el uso de ISS

(*Instruction Set Simulator*) que permiten simular la ejecución directa del código binario en los procesadores del sistema. Esta solución es muy precisa (proporcionando estimaciones a nivel de ciclo de reloj), pero es demasiado lenta para ser utilizada en primeras fases del proceso de diseño. Si el tiempo de simulación utilizando un ISS es un problema en la verificación de un sistema embebido con un procesador, cuando se intentan analizar redes de sensores inalámbricas con cientos/miles de nodos/procesadores la tarea se convierte en inviable. Por ello, se han propuesto nuevas técnicas de co-simulación que reducen el tiempo de simulación, como la virtualización [38] o la simulación nativa [39] [40]. Las técnicas de virtualización permiten convertir el código binario de la aplicación del nodo en código binario del ordenador en el que se ejecuta la simulación (*binary translation*). Esta conversión se realiza de forma dinámica durante la simulación. Las técnicas de simulación nativa (o compilada en el ordenador en donde se ejecuta la simulación) se basan en la instrumentación del código fuente del sistema a simular y su posterior compilación y ejecución en la plataforma en donde se realiza la simulación. La instrumentación introduce en el código fuente información sobre la plataforma HW, lo que permite estimar parámetros funcionales (comportamiento del RTOS del nodo) y no-funcionales (consumo y tiempo de ejecución).

2.3. Simulación de redes de sensores inalámbricas

2.3.1. Técnica de co-simulación HW/SW

Con el objetivo de simular distintos aspectos de las redes de sensores inalámbricas, se ha implementado un entorno de análisis de prestaciones basado en co-simulación nativa HW/SW. La utilización de metodologías de simulación nativa permite obtener resultados en tiempos más breves que utilizando un ISS,

pero con una precisión menor aunque más que suficiente para guiar el proceso de diseño. Dichas metodologías se basan en la instrumentación del código fuente del sistema a simular y su compilación en la plataforma en donde se realiza la simulación. Estás técnicas pueden utilizarse incluso cuando el desarrollo de la parte hardware no ha concluido, lo que es una importante ventaja sobre las técnicas basadas en virtualización, que exigen un detallado conocimiento del HW del sistema.

En este trabajo se presenta una herramienta de análisis de prestaciones basada en simulación nativa en la cual la ejecución del SW se simula en el ordenador (plataforma *host*) usando avanzados modelos abstractos del procesador, del RTOS (*Real Time Operating System*) y de la plataforma HW objetivo (plataforma *target*).

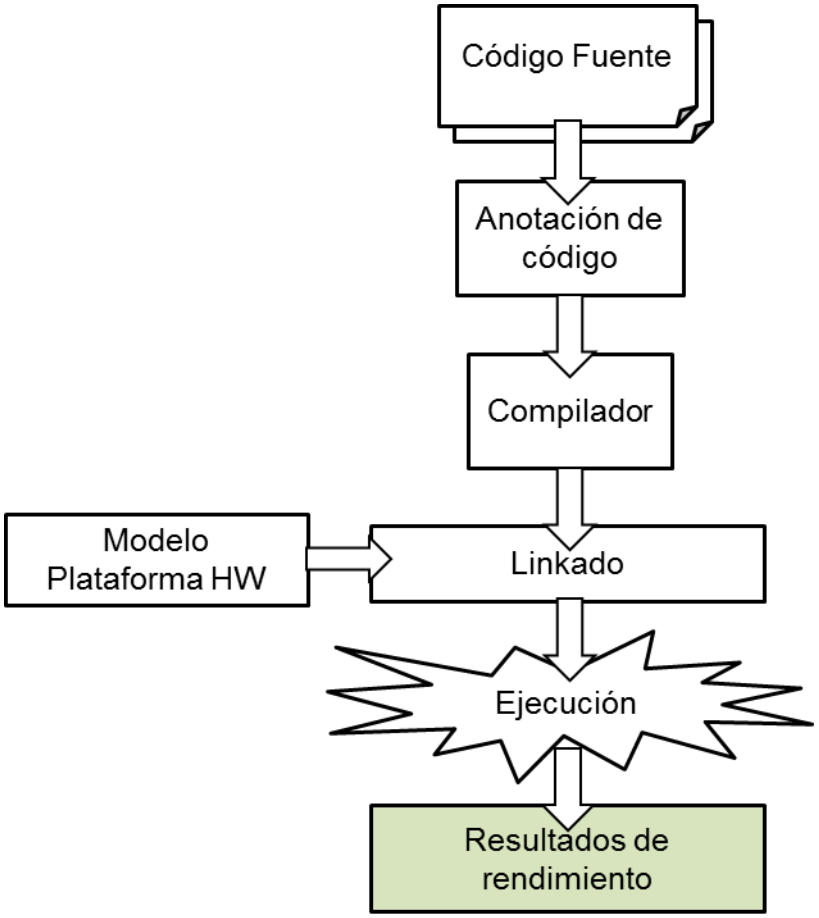


Figura 5: Proceso de co-simulación Nativa en la plataforma virtual

La metodología descrita en esta sección está basada en la técnica de simulación nativa [41] presentada en la Figura 5. Dicha metodología parte de la anotación (o instrumentación) del código fuente del software embebido con información relacionada con el hardware sobre el que se ejecuta (plataforma *target*). Para poder realizar la simulación, el sistema utiliza el núcleo del simulador del lenguaje SystemC. Este lenguaje es una librería C++ que permite describir sistemas hardware complejos. El simulador de SystemC es un programa de código abierto, lo que ha permitido utilizar su núcleo de simulación en nuestro entorno. Además, el uso de dicho núcleo también permite añadir componentes hardware SystemC a la simulación.

Gracias a las anotaciones introducidas en el código fuente de la aplicación, es posible obtener una estimación precisa y rápida de la energía consumida, tiempo de ejecución, número de accesos (*misses/hits)* a cachés de datos e instrucciones, número de accesos a buses, etc.

El proceso de compilación del código fuente conlleva 2 pasos:

1. El código fuente es analizado, siendo identificados los bloques básicos en el código. En cada bloque básico se añade una anotación con el coste (tiempo de ejecución, consumo, etc.) que cada bloque básico tiene en la plataforma en la que se va a ejecutar. Este proceso permite generar el código instrumentado.

2. El código instrumentado obtenido en la fase anterior es compilado nativamente en el ordenador en donde se realizará la simulación (plataforma *host*). La ejecución de dicho código simulará el comportamiento del sistema al tiempo que producirá estimaciones basadas en los costes añadidos (tiempo de ejecución, consumo, etc.).

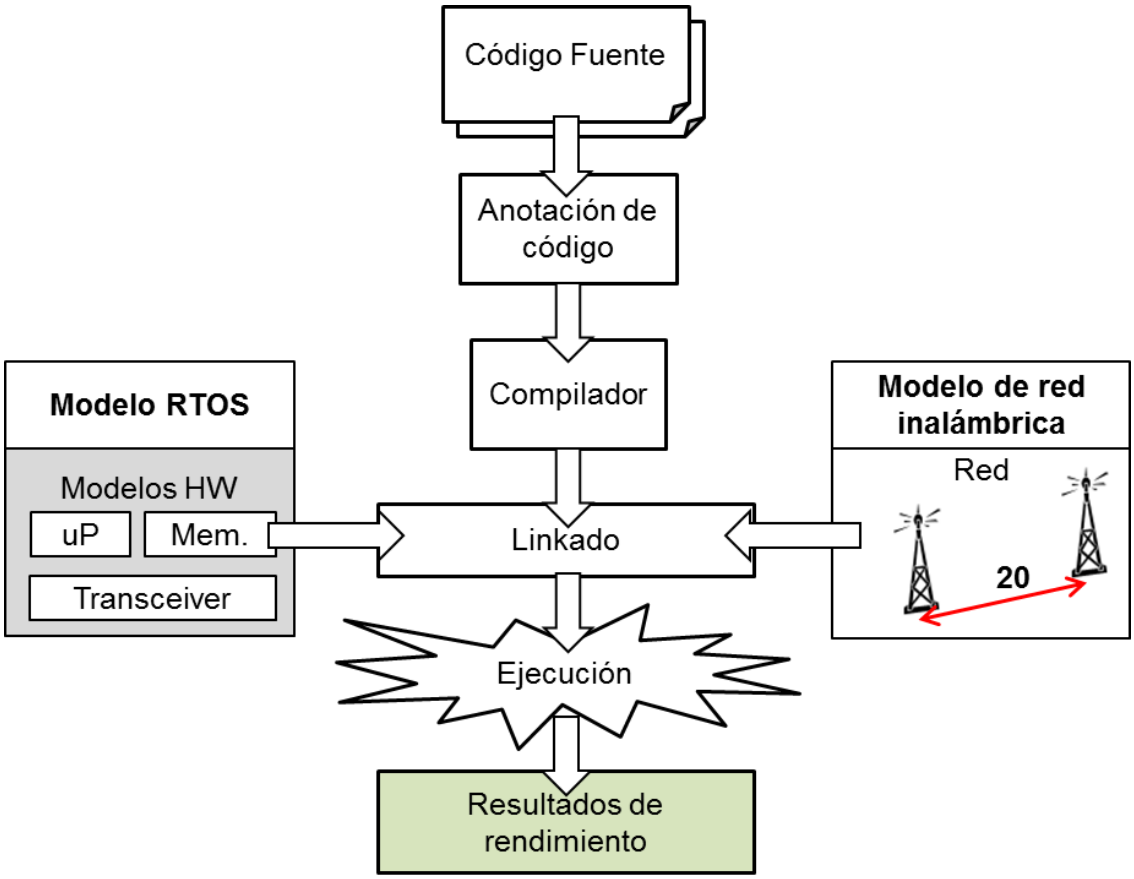


Figura 6: Proceso de co-simulación con las nuevas librerías

Adicionalmente, a este simulador se le añadirán distintas librerías (Figura 6) que se explicaran más detalladamente en las siguientes secciones. Dichas librerías incluyen modelos de simulación del RTOS y de la red, así como modelos de componentes HW típicos de las redes de sensores inalámbricas, como los receptores/transmisores de radio.

2.3.2. Modelado del Sistema Operativo de Tiempo Real

Un Sistema Operativo de Tiempo Real (RTOS) está destinado a proporcionar recursos del sistema a aplicaciones en tiempo real. Como Pronk explica en [42],

normalmente un RTOS crea y opera un cierto número de tareas de usuario concurrentes, planifica estas tareas garantizando una respuesta en un tiempo limitado usando mecanismos de prioridad y garantiza la no interferencia entre estas tareas y el núcleo (*kernel*) del RTOS. Además, permite la comunicación de las tareas mediante colas, su sincronización mediante *mutex* o semáforos, y la interacción directamente con el hardware a través de *drivers* de los dispositivos, relojes y mecanismos de interrupción.

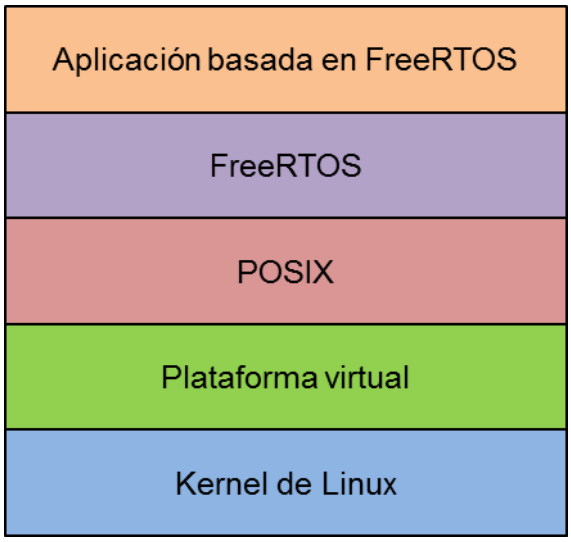


Figura 7: Modelado de FreeRTOS en el Simulador

La mayoría de las aplicaciones de los nodos están implementadas sobre un RTOS. Por esta razón, es muy importante modelar el sistema operativo en la plataforma de simulación. En [43] se presenta un estudio sobre los distintos RTOS que normalmente se utilizan en las redes de sensores inalámbricas. Para conseguir una mayor precisión en las estimaciones y poder simular sin modificaciones el código software introducido en los nodos es muy importante tener modeladas las funciones del Sistema Operativo. Una de las mayores ventajas de la metodología propuesta es el uso de la misma interfaz de programación (*Application Programming Interface*, API) que la plataforma física utiliza. Con este propósito, se ha utilizado un modelo basado en el API de POSIX [44] que se ha integrado en el entorno de análisis de prestaciones. El modelo que el simulador utiliza facilita la integración de nuevos RTOS, gracias a una implementación completa del API POSIX. Este modelo provee hilos (*threads*), *mutexes*, semáforos, colas, *timers* y

otros servicios comunes de POSIX. A partir de esta completa implementación de la infraestructura POSIX se puede modelar la mayoría de RTOS utilizados en redes de sensores inalámbricas. Como se puede ver reflejado en la Figura 7, la capa POSIX es soportada por el simulador desarrollado, que a su vez es ejecutado sobre el sistema operativo del *host* (Linux). Uno de los RTOS más populares en redes de sensores inalámbricas es FreeRTOS [45]. Para la integración de este RTOS en la plataforma virtual se ha colocado una nueva capa (Capa FreeRTOS) sobre la capa existente de POSIX. Gracias a esta capa, es posible simular aplicaciones desarrolladas para este Sistema Operativo. Esta capa implementa la funcionalidad de FreeRTOS utilizando funciones del API POSIX, como se puede observar en la Figura 7. El modelo actual implementa la versión V7.6.0 de FreeRTOS, la última disponible en el momento del desarrollo. En muchos casos, la implementación de las funciones con el enfoque adoptado únicamente requiere adaptar la interfaz de la API FreeRTOS para llamar a funciones similares de la API POSIX.

Además de modelar las funciones del sistema operativo, también es necesario modelar las funciones de acceso al hardware. El nodo tiene una serie de registros que permiten configurar y acceder a los distintos periféricos hardware de la plataforma *target*. Para acceder a ellos se suelen utilizar APIs específicas provistas por el fabricante del micro-controlador del nodo. Por ejemplo, la librería STM32LIB de STMicroelectronics proporciona una API común de acceso al hardware de todos los dispositivos de la familia STM32 de dicho fabricante. Dicho API permite portar el código embebido entre diferentes dispositivos de la misma familia sin tener que cambiar el código. Este API ha sido modelado en el simulador, lo que permite acceder al modelo de hardware desde el código de aplicación. Además, el simulador puede ejecutar el mismo código que será implementado en los nodos físicos y que usa dicha librería.

Adicionalmente, se ha modelado en el simulador el sistema operativo Windows, mediante la integración de la API de Win32 [44]. El modelado de este sistema operativo integra una virtualización de Win32 sobre la capa POSIX del simulador. El entorno de virtualización está provisto por WINE [46]. WINE es una aplicación de software gratuita que tiene como objetivo permitir a los sistemas operativos tipo Unix ejecutar programas escritos para Microsoft Windows. WINE implementa una

librería con la API de Win32, que actúa como puente entre la aplicación de Windows y Linux. La integración de esta API en la arquitectura del simulador es más compleja que en el caso anterior (FreeRTOS). Esto es debido a que, en este caso, la gestión de los manejadores de los objetos está realizada por WINE directamente.

2.3.3. Modelo de la red de sensores inalámbrica

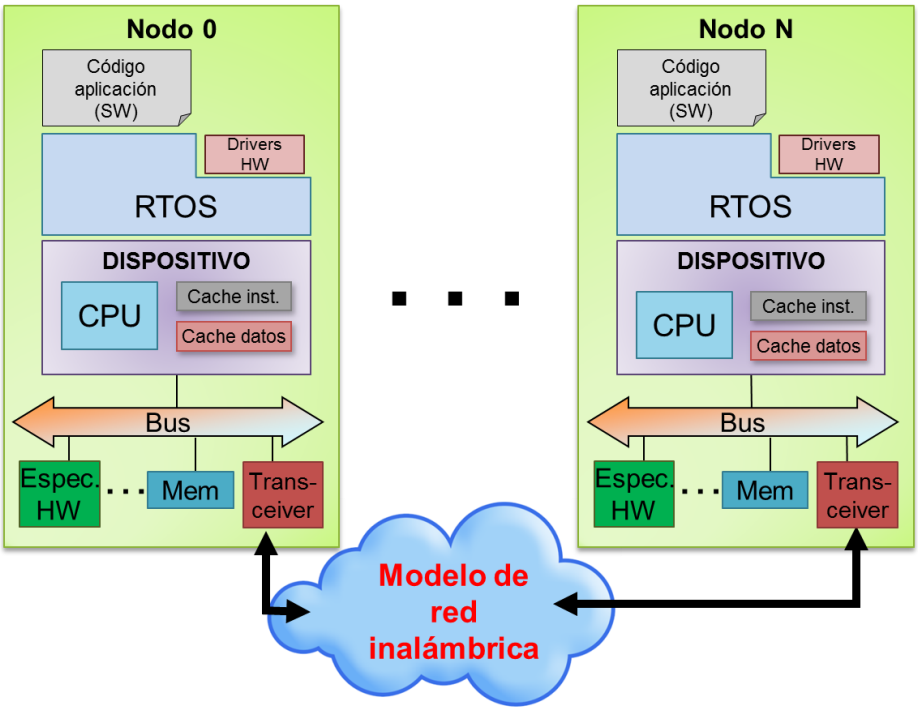


Figura 8: Representación de la arquitectura de los nodos + WSN

En la Figura 8 se muestra un modelo de red de sensores inalámbrica que incluye varios nodos (Nodo 1, …, Nodo N) y el modelo de red. Como se puede observar, la arquitectura HW y la aplicación de cada nodo es independiente de los otros nodos, lo que permite simular una red heterogénea con nodos con distinto hardware, firmware o incluso diferente sistema operativo. El software de aplicación de cada nodo ("Código aplicación (SW)" en la Figura 8) será instrumentalizado con

información correspondiente al nodo en el cual se ejecuta el código. El simulador proporcionará un modelo de RTOS (como por ejemplo FreeRTOS [47]) que permitirá abstraer detalles hardware de la plataforma objetivo (o *target*). Además, el entorno de simulación proporcionará un modelo de red de comunicaciones que será descrito en esta sección.

En una transmisión inalámbrica, el canal de comunicación entre nodos es el aire. Dicho canal es compartido por varios nodos y en él hay ruido e interferencias. Además, el alcance o distancia máxima entre dos nodos que se comunican es limitado. Como el canal es compartido, los mensajes pueden ser escuchados por nodos que no son los receptores de los mismos. Como consecuencia de todo esto, los desarrolladores deben determinar la visibilidad y la probabilidad de recepción correcta de un paquete enviado entre nodos (o la probabilidad de pérdida de dicho paquete). Con el objetivo de obtener simulaciones precisas, los desarrolladores deben disponer de información sobre la zona de despliegue de la red de sensores inalámbrica y definir una matriz con las probabilidades de pérdidas de paquetes entre nodos. Los valores de la matriz también deben incluir la probabilidad de pérdida de paquete debido al ruido en la red y equipos de radio. Estas probabilidades de pérdida de paquete pueden ser calculadas con simuladores electromagnéticos, como por ejemplo la herramienta Cindoor [48]. Gracias a Cindoor los desarrolladores pueden determinar la visibilidad de los nodos de una red, con sus rangos de alcance, y las probabilidades de error. Con la matriz de probabilidades de error, el simulador podrá conocer la efectividad de los enlaces entre nodos.

Un esquema de alto nivel del modelo de red se presenta en la Figura 9. Como se puede observar, la interfaz de red ("*hardware network interface*") es la responsable de transmitir/recibir los paquetes con destino/origen en otro nodo. Este modo de operación es muy similar al usado en un sistema real. En un sistema inalámbrico real, cuando un nodo envía un mensaje a otro, el paquete es transferido al transmisor de radio ("*transceiver*" de radio) para que lo envíe por radio frecuencia (RF). El mensaje no solo llega al receptor de radio del destino, sino a todos los nodos a los que tenga alcance. Será el receptor de radio a raves de su interfaz de

red el encargado de recibir los paquetes adecuados para que el firmware del nodo solo procese los mensajes que lo tienen por destino.

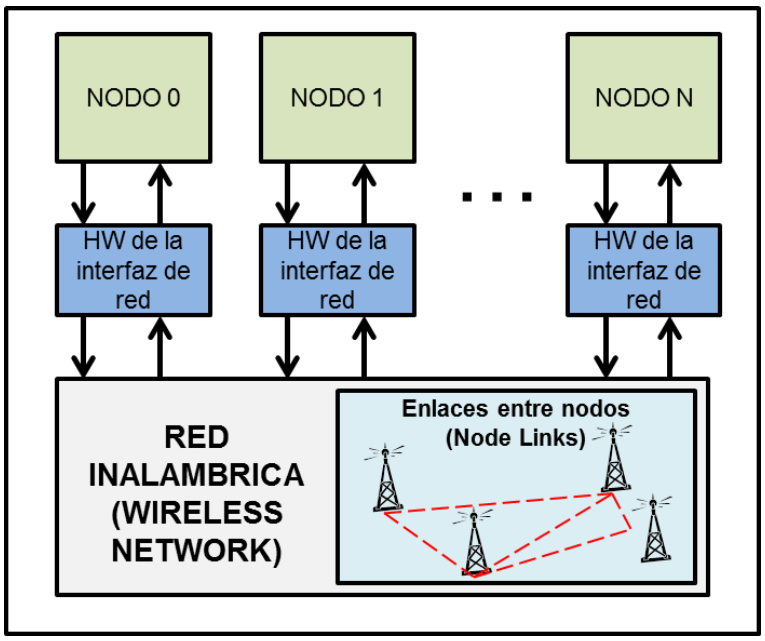


Figura 9: Modelo de la red

El modelo de red del simulador conoce el alcance de cada nodo y la probabilidad de recepción de mensajes a través de una matriz de porcentaje de error en la transmisión. Por ejemplo, si el desarrollador define el porcentaje asociada a un enlace ("*link*") entre dos nodos como 100, esto significará que el rango del nodo emisor no es lo suficiente potente como para alcanzar al nodo destino directamente (la probabilidad de pérdida de mensaje es del 100%). Sin embargo, si el desarrollador asocia al enlace un valor 0, la probabilidad de error en la transmisión será 0 y todos los paquetes enviados llegaran a su destino. Por ejemplo, en la Figura 10, el enlace entre el nodo 0 y el nodo 1 tiene asociado un valor del 10%, por lo que de cada 100 paquetes enviados a través de ese enlace (del nodo 0 al 1) solo 90 llegaran a su destino. Esta matriz la podemos ver reflejada en la Figura 10, identificando cada entrada de la matriz como “Enlace de red” o “*Node link*”.

	Nodo 0	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 0	0	10	0	0	12	7
Nodo 1	10	0	15	12	0	0
Nodo 2	0	15	0	9	0	17
Nodo 3	0	12	9	0	4	8
Nodo 4	12	0	0	4	0	5
Nodo 5	7	0	17	8	5	0

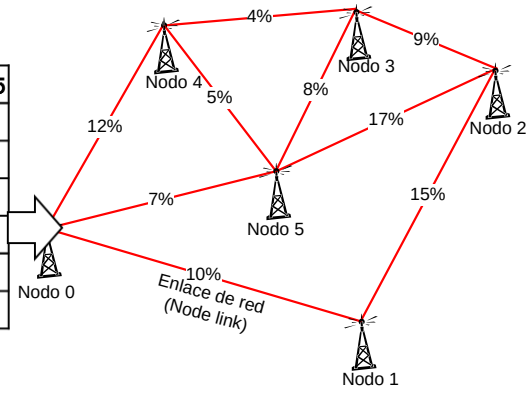


Figura 10: Representación de la red inalámbrica

El esquema del modelo de simulación de la red inalámbrica está representado en la Figura 11. El modelo de red es el responsable de transmitir los paquetes a su destino. Cuando un nodo envía un paquete, el modelo lo añade a su cola de mensajes que se ordena por el tiempo de llegada al nodo receptor. Cuando el tiempo de simulación coincide con el tiempo de llegada del paquete, el modelo de red inalámbrica saca el paquete de la cola y genera un número aleatorio entre 0 y 100. Si la probabilidad en porcentaje de recepción correcta ("*success*") es más grande que el número aleatorio, la red transmitirá el paquete a su destino. En otro caso, la red lo descartará. La probabilidad de recepción correcta ($P(success)$) se entiende como 100 menos el porcentaje de error ($P(error)$) definida en la matriz de enlaces de red.

$$P(success) = 100 - P(error)$$

Por ejemplo, en la Figura 10, un paquete enviado entre el nodo 0 y 1 será recibido únicamente en el caso de que el número aleatorio generado por el simulador sea mayor que el porcentaje de error definido en la matriz (10 en este caso). Sin embargo, en una red inalámbrica real, la transmisión se produce a través de un canal compartido y por ello el paquete será enviado a todos los nodos que están dentro de su rango (nodos 1, 5 y 4, con porcentaje menor de 100). Por esta razón, este paquete es enviado no únicamente a su destino en la simulación, sino también a los destinos para los que es visible el nodo emisor. En el caso de que ese paquete llegue a un nodo que no es su destino, será la interfaz de red y/o el software embebido el encargado de desecharlo. La interfaz de red identifica de

forma automática el destino del paquete y rechaza los mensajes que no son dirigidos al nodo, computando el coste en tiempo y consumo del proceso.

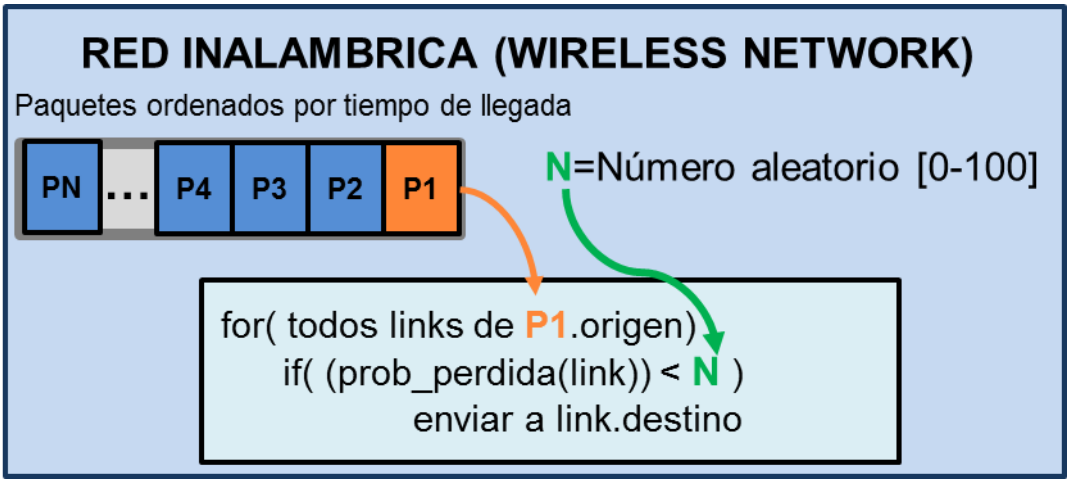


Figura 11: Esquema del funcionamiento de la red en el simulador

Este modelo de red permite la exploración del espacio de diseño teniendo en cuenta la topología de la red, sin la necesidad de recompilar el código cada vez que se modifican los parámetros el despliegue. Esto es posible porque la matriz de porcentajes de los enlaces se gestiona dinámicamente, lo que ayuda a reducir el tiempo de análisis de la red.

2.3.4. Modelado de componentes hardware específicos de un nodo

Hay componentes hardware en un nodo de red de sensores inalámbrica que normalmente otros sistemas no tienen. Estos componentes son el sensor y el módulo de radio ("*Transceiver*" de RF). El sensor es responsable de adquirir información del entorno con una cierta frecuencia o cuando un evento ocurra. Este componente se implementa en la simulación como un componente externo al software con un consumo especifico. Se pueden simular todos los tipos de sensores que se desee, difiriendo sus modelos únicamente en la información enviada al software, su consumo, tiempo de lectura y frecuencia de muestreo. El

otro componente fundamental es el módulo de radio. Dicho componente hardware es bastante más complejo que un sensor, integrando normalmente registros de configuración que controlan dinámicamente su modo de operación. Los registros implementados en el caso de uso de la siguiente sección modelan módulos de Digi [49] que usan el protocolo de radio XBee 802.15.4. Entre los registros implementados destacan los siguientes:

- **"*Destination Address High and Low*"**: Definen la dirección destino de un envío.
- **"*Baud Rate*"**: Especifican la velocidad de transferencia entre el módulo radio y el nodo (plataforma *target*).
- **"*Mac Retries*"**: Número de reintentos que se pueden utilizar para enviar un mensaje de tipo *unicast* (mensaje con un único destino). Si un mensaje no llega a su destino el módulo de RF lo reenviará hasta dicho número de veces, después de lo cual considerará que la transmisión es imposible/errónea.
- **"*Multiple Transmissions*"**: Número de transmisión en un envío *broadcast* (mensaje con múltiples destinos). Un envío *broadcast* no necesita confirmación y el nodo receptor no envía confirmación de que el paquete haya llegado a su destino.
- **"*Power Level*"**: El nivel de potencia a la cual el módulo RF transmite.

La modificación de los registros del módulo de radio durante la simulación se realiza utilizando los mismos comandos AT que se utilizan en el hardware real. Para conseguir esta transparencia entre el modelo de simulación y el real se ha implementado una API que permite interactuar con los registros del HW de la misma manera que en una aplicación real.

Con el fin de poder modelar el consumo de energía con precisión, es importante caracterizar correctamente el módulo de radio (*transceiver*). Un ejemplo de las

medidas obtenidas se puede observar en la Tabla 1, donde se muestra el consumo de un módulo XBee Pro. Como se puede apreciar, su consumo depende del estado en el que se encuentra el módulo. En caso de que el estado sea de transmisión, su consumo dependerá de la configuración de potencia, la cual puede ser configurada desde 10dBm a 18dBm en pasos de 2dBm. Todos estos parámetros han sido introducidos en el modelo de simulación y utilizados durante la estimación de consumo del sistema.

Tabla 1: Medidas de consumo del XBEE PRO

STATE	POWER LEVEL	DATA
Transmisión	10 dBm	138.31 mA
	12 dBm	156.45 mA
	14 dBm	170.92 mA
	16 dBm	189.12 mA
	18 dBm	216,40 mA
Recepción	Indiferente	56.52 mA
Espera (Idle)		54.79 mA
Power-down		39.12 μA

2.3.5. Generación de informes de la simulación

Las aplicaciones software que se ejecutarán en los nodos de la red de sensores inalámbrica están escritas en C/C++, siendo compiladas con el compilador del simulador. Para poder compilar el código, es necesario especificar la plataforma hardware en la cual se ejecutará dicho código. Además, se requieren librerías y funciones de estimación que modelen el hardware y RTOS de cada nodo. Por supuesto, también será necesario definir la topología y características de la red inalámbrica. Una vez que todos estos parámetros han sido procesados, el entorno genera un modelo ejecutable: la plataforma virtual o simulador. Cuando se ejecuta

este modelo, se podrán obtener las siguientes estimaciones para cada nodo de la red:

- **RTOS:** Datos del Sistema operativo.
  - *Processes*:
    - *Created*: Número de procesos creados.
    - *Destroyed*: Número de procesos destruidos.
    - *Mean process duration:* Tiempo del proceso principal.
  - *User Time*: Tiempo total de usuario.
  - *Kernel Time*: Tiempo total de *Kernel*.

- ***Transceiver***: Datos de la interfaz de red.
  - *Energy Standby*: Energía del *transceiver* en modo *standby* (en Julios).
  - *Energy transmission*: Energía del *transceiver* durante las transmisiones (en Julios).
  - *Power*: Potencia del *transceiver* in Watts.
  - *Packets send*: Número de paquetes enviados.
  - *Packets received*: Número de paquetes recibidos.

- **Processor**: Datos de cada procesador del sistema.
  - *Number of switches:*
    - *Thread switches*: Número de cambios de hilo o *thread*.
    - *Context switches:* Número de cambios de contexto.
  - *Time estimations:*
    - *Running time*: Tiempo en ns de la CPU corriendo.
    - *Use of CPU*: porcentaje de uso de la CPU.
  - *Instructions:*
    - *Executed instructions:* Número de instrucciones ejecutadas.
    - *Cache misses:* Número de *misses* en la caché de instrucciones
  - *Data:*

        - *Cache misses:* Número de *misses* en la caché de datos.
    - *Energy and Power:*
        - *Core:* Energía y potencia del procesador en Julios y Watios, respectivamente.
        - *Instruction cache*: Energía y potencia de la caché de instrucciones en Julios y Watios, respectivamente.
        - *Data cache:* Energía y potencia de la caché de datos en Julios y Watios, respectivamente.

- **Simulation time:** Tiempo de simulación en segundos.

2.4. Simulación de redes de sensores: Caso de uso

En esta sección se va a evaluar con un ejemplo real el simulador descrito en los apartados anteriores. Dicho simulador utilizará los módulos presentados hasta ahora: modelo de red, modelo de sistema operativo y modelo de componentes HW específicos de las redes inalámbricas. El próximo capítulo extenderá este simulador para permitir la simulación de ataques.

2.4.1. Escenario a simular

El objetivo del sistema a diseñar es la monitorización periódica del estado de unos depósitos con fines medioambientales. La red de sensores incluye tres tipos de nodos: *Gateway*, Repetidor y Nodo Sensor. Cada tipo de nodo tiene una arquitectura hardware distinta y un software diferente.

El primer tipo de nodo es el *Gateway*, que sirve de enlace entre los nodos de la red y el exterior. Este nodo es responsable de comunicarse con el segundo tipo de nodo: el Repetidor. El Repetidor sirve de enlace entre nodos de la red y está a

cargo de la comunicación con el tercer tipo de nodo: los denominados “Nodos Sensores”. Cada nodo de este tipo tiene un sensor que adquiere información del agua de un depósito. Cuando el nodo lee la información del sensor, la envía al Repetidor, que a su vez la envía al Gateway. El Gateway espera a recibir la información de todos los sensores para componer un mensaje con todos los datos de todos los sensores para enviarlo a través de un módulo GPRS.

Cuando un nodo finaliza su funcionalidad, pasa a un estado de bajo consumo o modo *sleep*. Esto permite aumentar el ciclo de vida de la red.

2.4.2. Modelo de la red del caso de uso

La Figura 12 describe el esquema del modelo de la red con 9 nodos interconectados. Las líneas rojas representan las conexiones entre los nodos que tienen comunicación real. Las líneas discontinuas representan los nodos que se pueden ver (reciben mensajes entre ellos), pero entre los cuales no hay comunicación directa (no hay mensajes enviados de un nodo de la línea discontinua al otro).

En este ejemplo, la probabilidad de error en la transmisión de los paquetes está basada en los datos capturados en el despliegue de un sistema real. En otros ejemplos en donde no estarían disponibles datos con esta precisión, se podría definir cada enlace con la probabilidad que considere el usuario o con estimaciones realizadas mediante programas como Cindoor [48]. En esta caracterización de canal es importante incluir las probabilidades de interferencia causadas por la acumulación de nodos en un espacio limitado, así como tener en cuenta los posibles obstáculos que pudieran existir en la comunicación. En el caso de uso, después de completar el despliegue, se analizó el tráfico de la red. Con estos datos, se puede observar que cada enlace tiene una tasa de error distinta. Esto era previsible ya que cada nodo tiene una ubicación distinta. La tasa de error de transmisión va desde el 2%, en el mejor de los casos, al 67% en el peor caso. Estas tasas están representadas en la Figura 12.

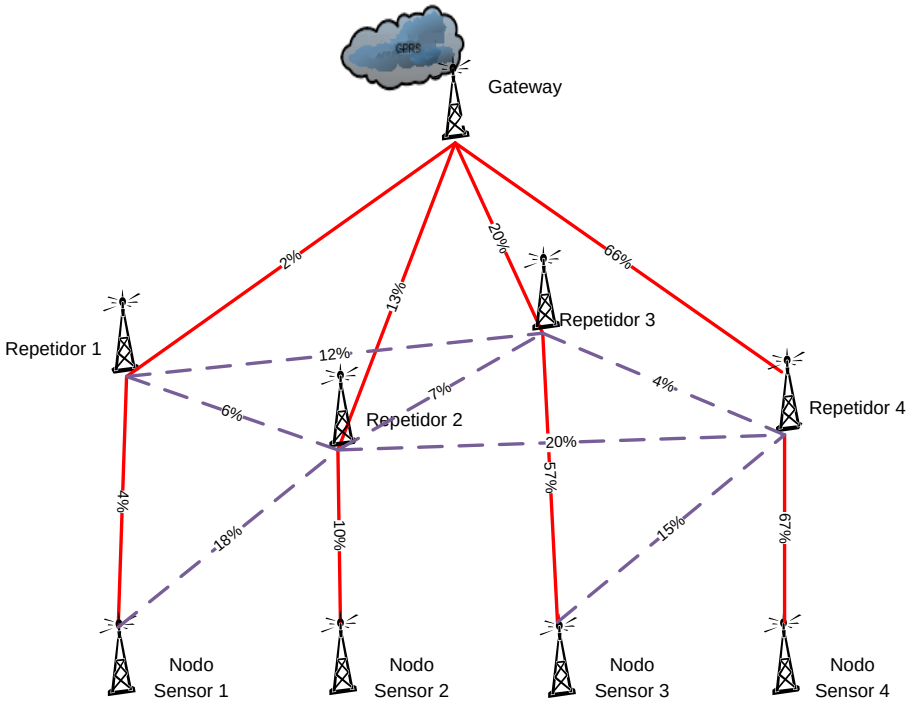


Figura 12: Representación del modelo de red a estudiar

2.4.3. Modelo de la plataforma hardware

Esta sección describe los modelos hardware de cada nodo de la red. Todos los nodos comparten varios componentes hardware: un *transceiver* XBee 802.15.4 y un microprocesador Cortex M3 [50] de ARM, corriendo a 90 MHz, con su memoria y sus periféricos. Además, cada nodo incluye componentes hardware específicos:

- **Gateway**: El módulo GPRS. Este componente se modela con un consumo de energía promedio por estado de operación y un tiempo asociado a cada mensaje procesado.

- **Nodo Sensor**: El nodo integra al sensor responsable de tomar las medidas medioambientales. Este componente esta modelado como un sensor genérico, que lee información del ambiente cuando el sistema lo requiere.

- **Repetidor**: No tiene componentes hardware adicionales. Su función es servir de enlace entre los dos tipos de nodos previos.

El componente principal de todos estos nodos es el *transceiver* de radio XBee. Como se ha comentado anteriormente, el modelo incluye los registros de configuración para controlar su modo de operación mediante comandos AT. En el ejemplo se usa la configuración por defecto de XBee Pro. Los valores de los parámetros del módulo de radio más importantes son:

- *Baud Rate*: 9600bps
- *Mac Retries*: 10
- *Multiple Transmissions*: 0x3
- *Power Level*: 4 o 18dBm.

2.4.4. Software de aplicación

Esta sección describe el software (*firmware*) que se ejecuta en cada tipo de nodo. Como se comentó previamente, cada tipo de nodo ejecuta un programa distinto, ya que cada uno tiene su propia funcionalidad. Todos los nodos utilizan el mismo sistema operativo: FreeRTOS. A continuación se describe brevemente cada aplicación software:

- **Gateway**: El Gateway es el responsable de enviar un mensaje a los nodos repetidores para verificar que están activos (no están en modo *sleep*, dormidos). Cuando todos los repetidores están activos, el Gateway espera respuesta de todos los nodos con los datos de lectura de los sensores de los depósitos. Una vez que el Gateway ha recibido la información de todos los nodos, compone un mensaje y lo envía a Internet usando el módulo GPRS. Después de esto, la aplicación envía una interrupción para que el nodo pase a estado dormido (*sleep*) durante 10 minutos

- **Repetidor:** Cuando el repetidor se despierta, espera recibir un mensaje del Gateway para, a su vez, enviar un mensaje al Nodo Sensor. Si el Nodo Sensor responde con los datos del depósito, el repetidor envía esta información al Gateway y pasa a estado inactivo (dormido) durante 10 minutos.

- **Nodo Sensor**: La función de este nodo es esperar un mensaje del repetidor. Después de recibirlo, lee información del agua en el depósito usando su sensor y envía la información al Repetidor. Cuando finaliza su función, este nodo pasa a estado dormido (*sleep*) durante 10 minutos. Por lo tanto, la frecuencia de adquisición de datos es de un dato cada 10 minutos.

2.4.5. Resultados de la simulación

Para validar los resultados del simulador, se han medido físicamente y simulado la red en dos entornos distintos. La primera prueba se ha realizado en el laboratorio bajo condiciones casi “ideales”. La segunda prueba se ha realizado con la red desplegada en el entorno real.

En el primer caso, las aplicaciones de los nodos se han ejecutado al tiempo que se medía el consumo de cada nodo. Los nodos estaban en un laboratorio, con una tasa de error en la transmisión cercana a 0, ya que la distancia entre los nodos era pequeña y las condiciones en el laboratorio casi ideales. En el laboratorio, los receptores recibían todos los paquetes sin necesidad de reintentos. Por ello, se definió una red con una tasa de error del 0%, a partir de la cual se obtuvieron estimaciones mediante simulación. Las comparaciones entre las medidas reales tomadas en el laboratorio y las estimaciones obtenidas con el simulador se presentan en la Tabla 2, Tabla 3 y Tabla 4. La Tabla 2 muestra los datos del nodo “Nodo Sensor”. La Tabla 3 muestra las mediciones y estimaciones del nodo “Repetidor” y la Tabla 4 proporciona los mismos parámetros pero para el Gateway.

En estas tablas se puede ver que los errores de las estimaciones están entre el 5% y el 8%.

Tabla 2: Comparación de consumos del Nodo Sensor

	Componente	**Modos**		**TOTAL**
Consumo Simulado	*Transceiver*	*Stand-By*	0.11352 J	0.12735 J
		Transmi.	0.00915 J	
	Core+Caches	0.00467 J		
Datos reales medidos	Sistema Completo	0.13361 J		0.13361 J
			ERROR	4.919%

Tabla 3: Comparación de consumos del Gateway

	Componente	**Modos**		**TOTAL**
Consumo Simulado	*Transceiver*	*Stand-By*	0.20627 J	0.51026 J
		Transmi.	0.02447 J	
	Core+Caches	0.27959 J		
Datos reales medidos	Sistema Completo	0.54363 J		0.54363 J
			ERROR	6.538%

Tabla 4: Comparación de consumos del Repetidor

	Componente	**Modos**		**TOTAL**
Consumo Simulado	*Transceiver*	*Stand-By*	0.08643 J	0.10660 J
		Transmi.	0.01525 J	
	Core+Caches	0.00491 J		
Datos reales medidos	Sistema Completo	0.11481J		0.11481J
			ERROR	7.702 %

En el segundo caso, se ha evaluado la red desplegada en el escenario real y que fue descrito en la primera parte de esta sección. Para ello, se ha simulado una hora de funcionamiento de la red real y, con los datos obtenidos, se ha realizado una comparación entre los repetidores 1 y 4 (Figura 13) y los nodos Sensores 1 y 4 (Figura 14). Los nodos sensores y repetidores de dichas figuras aparecen en la Figura 12. Los datos de la Figura 13 y la Figura 14 muestran como el consumo de los nodos se incrementa debido al incremento de la tasa de error de los enlaces. La Figura 13 muestra un incremento en el consumo de alrededor del 15%. La Figura 14 muestra un incremento más comedido, de alrededor de un 6%, lo que también puede ser un incremento crítico. Al tratarse de nodos idénticos, con la misma arquitectura hardware y software, esta variación del consumo puede ser un problema muy serio. Gracias a la simulación, se ha podido detectar que el Repetidor 4 es un nodo problemático, siendo muy probable que su ciclo de vida sea menor que el resto de los nodos de la red. Existen varias soluciones para este problema: desde buscar una nueva ubicación para este nodo, hasta proveerlo con una batería con mayor capacidad pasando por modificar su software específico para que envíe los mensajes con más potencia y, de esta forma, se reduzca el número de reintentos.

Otra comparación entre distintos tipos de nodos se presenta en la Figura 15, donde se puede apreciar que el nodo más crítico, teniendo en cuenta su consumo, es el Gateway.

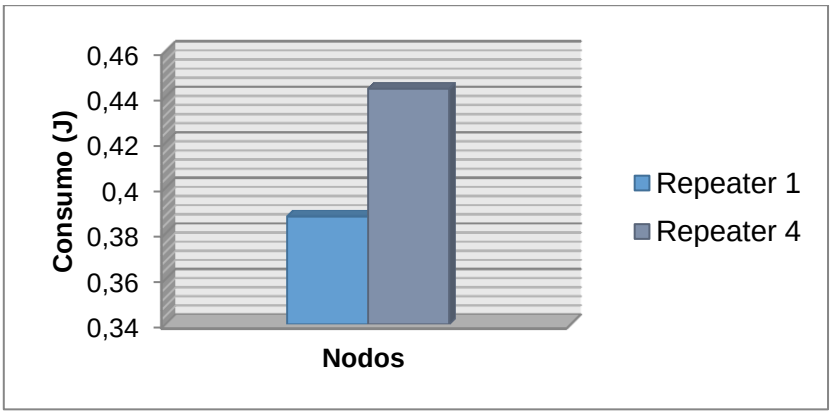


Figura 13: Consumo simulado de los Repetidores 1 y 4

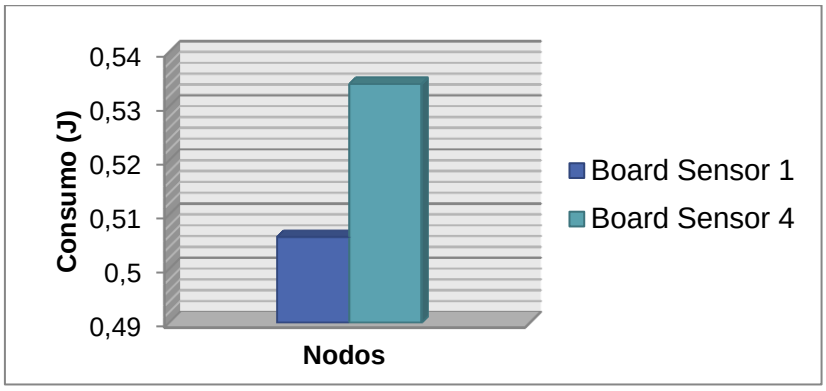


Figura 14: Consumo simulado de los Nodos Sensores 1 y 4

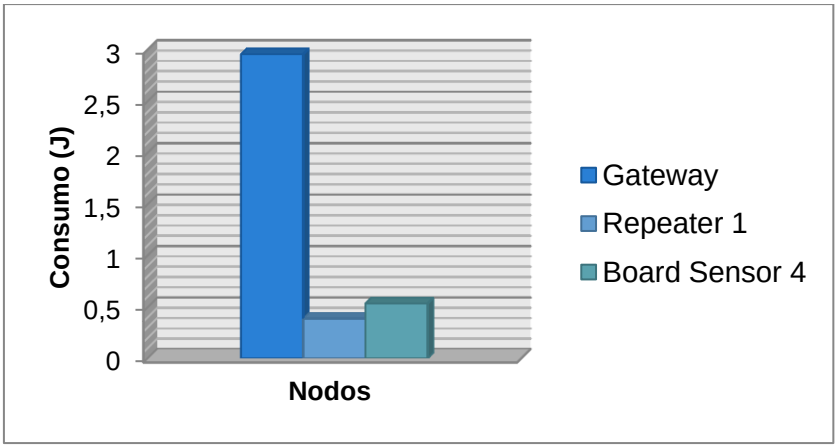


Figura 15: Comparacion entre distintos tipos de nodos

Por razones de confidencialidad, no se han podido introducir los datos de consumo del sistema en la red desplegada debido a que la red simulada en esta sección es un sistema industrial que ha sido desplegado en un entorno real.

3. Análisis de redes de sensores seguras

En el capítulo anterior se ha presentado la metodología desarrollada para simular y explorar distintas alternativas de diseño de redes de sensores inalámbricas bajo condiciones normales. Sin embargo, como se ha mencionado previamente, en la mayoría de los casos las redes son desplegadas en entornos hostiles, lo cual aumenta el riesgo de ataques y violaciones de la seguridad al estar situados los nodos en lugares vulnerables. Por lo tanto, es importante que durante la especificación y el proceso de diseño de las redes los desarrolladores tengan en cuenta un amplio abanico de eventos inesperados que pueden impactar en el correcto funcionamiento de la red así como los efectos que estos provocan. Para hacer frente a estos riesgos, se ha integrado en el simulador modelos de atacantes externos, lo que permite replicar los efectos de distintos ataques en el funcionamiento de la red.

3.1. Seguridad de sistemas en red

Durante los últimos años, existe una preocupación creciente por la seguridad de los sistemas embebidos en red y su vulnerabilidad en caso de ataques. Por ello, se están empezando a introducir cada vez más requisitos de seguridad en la especificación de las redes inteligentes. Debido al tipo de información que manejan estos dispositivos y al tipo de despliegue que tienen estas redes (normalmente un despliegue inalámbrico y

desatendido), es necesario mejorar los mecanismos de seguridad actualmente utilizados. Aunque algunos de los riesgos de seguridad conocidos en las redes tradicionales son aplicables a las redes de sensores inalámbricas, estas últimas tienen vulnerabilidades específicas. Dichas vulnerabilidades son debidas a características propias de estas redes como su canal de comunicación (inalámbrico, compartido, vulnerable y desprotegido), el despliegue en entornos hostiles y desatendidos, su capacidad computacional reducida, su estricto manejo de la energía y la propia complejidad del sistema. Además, los atacantes tendrán normalmente acceso directo al hardware de cada nodo, lo que les proporciona un gran número de posibles vías para atacar los dispositivos o la red completa. Debido a estas características, las redes de sensores inalámbricas son especialmente susceptibles a recibir un gran número de ataques contra su seguridad [51]. En los últimos años, un requisito muy común de las redes de sensores inalámbricas es la valoración de la seguridad, especialmente debido a que los atacantes pueden corromper la red, accediendo o modificando la información y afectando al comportamiento de los nodos.

El nivel de seguridad requerido puede variar de una aplicación a otra dependiendo de la importancia de la información que se va a obtener y/o intercambiar [52]. Por esta razón, es importante identificar y tener en cuenta las debilidades de la red inalámbrica en las primeras fases del diseño. Además, conocer los efectos potenciales de los ataques típicos a un nodo o a una red facilitará prevenir otras vulnerabilidades. Es por ello que el conocimiento de las consecuencias de los ataques más típicos es un gran valor añadido en el diseño del Hardware y del Software embebido en los nodos y de la red. Por lo tanto, la posibilidad de simular ataques es un valor añadido muy importante cuando se desarrolla el software de los nodos o la configuración de la red inalámbrica.

Debido a los requisitos de bajo coste y bajo consumo que tienen este tipo de sistemas, los nodos tienen normalmente limitaciones tanto en recursos hardware/software como en capacidad de cómputo. Es por esto que el diseño de la seguridad en redes de sensores inalámbricas tiene que tener en cuenta la memoria, capacidad computacional y disponibilidad de recursos hardware/software. Además de esto, el consumo de energía está limitado. Normalmente el consumo de potencia es la mayor restricción que tienen los diseñadores cuando desarrollan redes de sensores inalámbricas. Los nodos suelen ser dispositivos alimentados mediante baterías, por lo

que la vida de la batería limita la vida de los nodos y de la red. Como ya se ha comentado, estos nodos suelen estar en entornos hostiles y desatendidos donde el acceso físico a los nodos después del despliegue es complicado por lo que el reemplazo de las baterías es muy difícil y costoso. Además, si tenemos en cuenta el alto número de nodos con los que pueden contar estas redes y la distribución de los mismos en la red, el coste de reemplazar las baterías aumentaría tanto que en la mayoría de los casos no compensaría. Por todo ello, es esencial tener en cuenta el impacto en el consumo de las medidas y/o estrategias utilizadas para aumentar la seguridad del sistema. Además, hay que tener en cuenta que uno de los principales efectos que tienen los ataques a las redes de sensores inalámbricas es el incremento del consumo en los nodos atacados, lo cual reduce la vida útil del nodo y de la red. Por todo ello, es esencial estimar el impacto en consumo tanto de los ataques como de las estrategias que se adopten para mejorar la seguridad del sistema.

En esta sección se propone ampliar la plataforma presentada en la sección 2 (Simulador de redes de sensores inalámbricas mediante simulación nativa) con la capacidad de simular ataques. Dicha ampliación permitirá determinar los ataques más perjudiciales y ayudará a diseñar y/o implementar contramedidas que ayuden a detectar y evitar estos riesgos de seguridad. Este enfoque es muy eficaz ya que se aplica antes del despliegue de la red, durante la fase de diseño y desarrollo del software y del hardware del sistema. Esto permite a los desarrolladores diseñar sistemas más seguros ayudándoles a entender el comportamiento de la red bajo múltiples condiciones e introduciendo contramedidas que eviten los efectos de los ataques.

En este trabajo, el primer paso es identificar los ataques conocidos más dañinos que se producen en una red. El segundo paso será definir tres tipos de atacantes que permitirán modelar todos los ataques identificados. Las principales contribuciones de este apartado respecto al estado del arte son:

- Se presenta una metodología que permite el diseño de redes de sensores inalámbricas seguras desde las primeras fases del proceso de diseño. Esta metodología usa un simulador para identificar los ataques más peligrosos que la red inalámbrica puede sufrir. Además el simulador facilita el desarrollo y la verificación de distintas medidas contra ataques.

- Se presenta un modelo de ataque que clasifica un gran número de ataques en únicamente cuatro categorías, lo cual facilita su implementación y análisis. Este modelo de ataque se integrará en el simulador desarrollado, el cual tiene en cuenta la topología de la red, la plataforma HW/SW de cada nodo y el código software de la aplicación real que se ejecuta en el procesador.

- Por último, el simulador incluye una métrica llamada "SEM", que será presentada en el próximo capítulo y que permite la evaluación de la seguridad de las transmisiones.

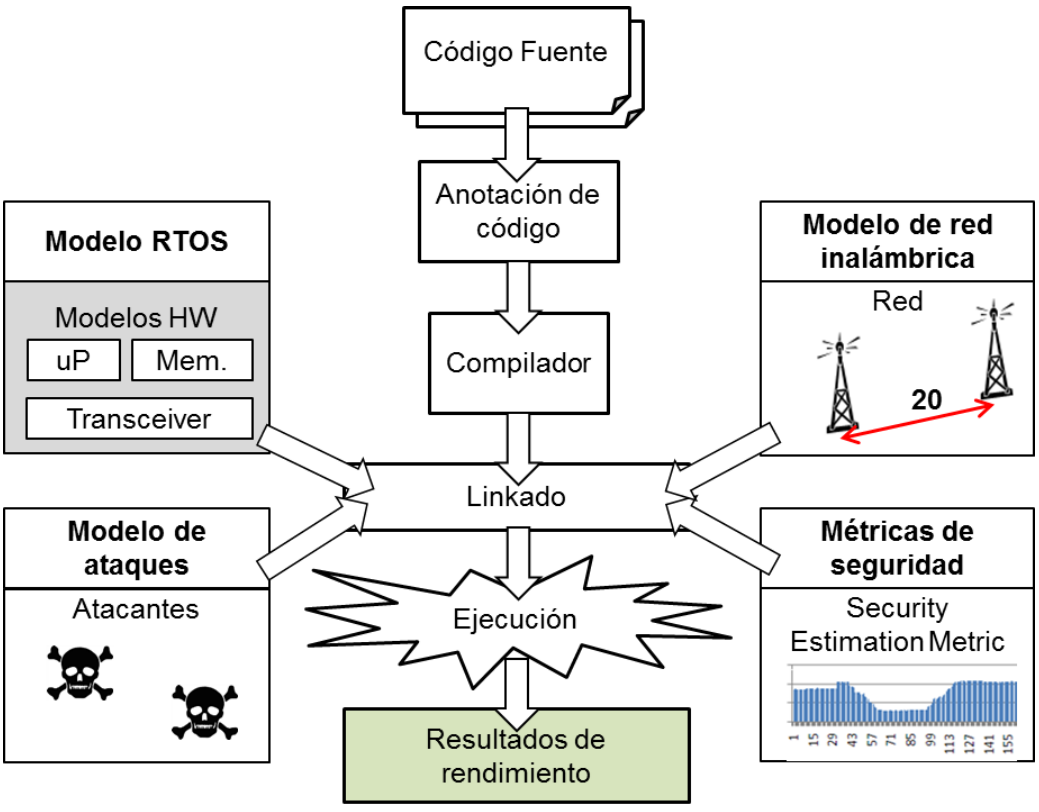


Figura 16: Proceso de co-simulación con las nuevas librerias de seguridad

El simulador de ataques permite modelar, simular y estimar el impacto de los ataques con distintas configuraciones de la red inalámbricas. Gracias a esta estimación del comportamiento del nodo atacado es posible diseñar e implementar un software que evite los ataques. La Figura 16 presenta el flujo del simulador de redes seguras que

incluye el modelo de los ataques que se presenta en esta sección. También se incluye la métrica de seguridad que se presentará en el capítulo 4. El entorno de simulación de redes seguras es una extensión del simulador presentado en la Figura 6, como se puede observar.

La seguridad afecta a un elevado número de tareas en la red inalámbrica, desde autentificación de nodos, a integridad de mensajes, privacidad y no-repudio de información. Cuando se define la seguridad en redes de sensores inalámbricas hay una serie de términos que conviene clarificar[53]:

- ***Confidentiality*** **(Confidencialidad):** La confidencialidad es uno de los problemas más importantes relacionados con la seguridad en redes [54] [55]. La confidencialidad busca que un mensaje dado no pueda ser entendible por nadie que no sea el destinatario.

- ***Integrity*** **(Integridad):** Conseguir la confidencialidad implica que la información no puede ser robada. Sin embargo no implica que la información sea segura. El atacante podría cambiar la información obligando a la red a trabajar con información no genuina. La integridad certifica que un mensaje enviado de un nodo a otro no ha sido modificado por un nodo maligno.

- ***Freshness*** **(Frescura):** Incluso cuando se garantiza la integridad y confidencialidad de los datos, la red sigue siendo vulnerable. El riesgo nace del uso, por parte del nodo atacante, de datos antiguos y no actualizados. La frescura implica que los datos son recientes y se asegura de que ningún atacante pueda reproducir los mensajes antiguos.

- ***Availability*** **(Disponibilidad**): Se asegura de que los servicios de la red siempre estén disponibles.

- ***Authorization*** **(Autorización):** Se asegura que solo los nodos autorizados puedan proveer información a la red.

- ***Authentication* (Autenticación):** Se segura que la comunicación entre nodos sea genuina, esto es que un nodo malicioso no pueda enmascare como un nodo real de la red.

- ***Non-repudiation* (No-repudio):** Se refiere a que un nodo no pueda rechazar la retransmisión de un mensaje que se ha enviado previamente por un nodo genuino.

Un ataque puede ser definido como un intento de obtener acceso no autorizado a un servicio, recursos o información de la red. Como consecuencia de ello, comúnmente el ataque compromete la integridad, disponibilidad o confidencialidad del sistema. La metodología propuesta se ha desarrollado para modelar estas consecuencias, facilitando la evaluación y la exploración de distintas alternativas para evitar o minimizar el impacto de los ataques.

3.2. Simulación de redes de sensores seguras: Estado del arte

Para poder estimar las prestaciones del sistema y evaluar su seguridad, es importante tener un simulador que reúna los siguientes requerimientos:

- **Uso del tráfico real de la red.** Para estimar los efectos de los ataques o la seguridad de los mensajes enviados es muy importante poder trabajar con el tráfico real en la red desplegada. No es posible utilizar estimaciones o tráfico aleatorio en la estimación de la seguridad.

- **Simulación del código SW real**. Para evaluar el comportamiento de SW cuando el nodo está siendo atacado o sus mensajes modificados es muy importante simular la red usando el código software real que se ejecuta en cada nodo.

- **Soporte para la plataforma HW**. El consumo de potencia depende de los componentes HW de la plataforma. Es importante que el simulador permita utilizar y definir diferentes arquitecturas.

- **Soporte para el Sistema Operativo.** El RTOS tiene un papel esencial en el comportamiento del sistema. Por ello es importante que el simulador soporte distintos RTOS de forma eficiente.

- **Consumo de potencia.** Uno de los principales efectos que buscan la mayoría de los ataques es el incremento del consumo de potencia para, de esta forma, eliminar a un nodo genuino de la red. Por lo tanto, es esencial poder estimar el consumo de cada nodo tanto en funcionamiento normal como cuando el nodo es atacado.

En el capítulo anterior se analizaron varios entornos de simulación de redes de sensores. Sin embargo, dichos entornos no cumplen muchas de las condiciones anteriormente comentadas, como se muestra en las tablas adjuntas:

Tabla 5: Estudio de simuladores

Simulador	NS-2	NS-3	TOSSIM	UWSim	Avrora
Generación del tráfico	Patrones de tráfico	Patrones de tráfico	Estáticamente o dinámicamente	Dinámicamente	Real
Soporte para SW Real	NO	NO	Solo TinyOS	NO	SI
Plataforma HW	NO	NO	NO	SI	Limitada
Soporte SO	NO	NO	TinyOS	NO	NO
Consumo de potencia	SI	SI	Con PowerTOSSIM	NO	SI
Limitaciones	No tiene tráfico real	No tiene tráfico real	Solo para código sobre TinyOS	Solo para redes submarinas	Solo para nodos Mica2

Tabla 6: Estudio de simuladores (Continuación)

Simulador	Castalia	GloMoSim	Shawn	J-Sim	Prowler
Generación del tráfico	Real	Estadístico	No real	No real	Probabilístico
Soporte para SW Real	SI	NO	NO	NO	NO
Plataforma HW	NO	NO	NO	NO	MICA Mote
Soporte SO	NO	NO	NO	NO	TinyOS
Consumo de potencia	SI	NO	NO	SI	NO
Limitaciones	No tiene soporte HW	Tráfico estadístico, no tiene modelos de energía	Tráfico no real	Eficiencia baja y tráfico no real	Tráfico probabilístico.

Tabla 7: Estudio de simuladores (Continuación)

Simulador	ATEMU	OMNeT++	COOJA	Desarrollado
Generación del tráfico	Real	Eventos	Real	Real
Soporte para SW Real	SI	NO	SI	SI
Plataforma HW	Sistemas basados en procesador AVR	SI con extensiones	SI	SI
Soporte SO	TinyOS	NO	Contiki OS.	Múltiples
Consumo de potencia	SI	SI	SI	SI
Limitaciones	Solo para Sistemas basados en procesador AVR	Lento y no soporta el código SW real	Eficiencia baja y número limitado de distintos tipos de nodos simultáneos.	

La principal carencia de estos simuladores de redes es que ninguno de ellos tiene un nivel realista de simulación con diferentes sistemas operativos y con estimación de consumo. Otro problema de estos entornos de simulación es que no permiten la ejecución del SW real, por lo que el tráfico de la red está basado en funciones externas, las cuales no pueden generar el tráfico que tendría la red realmente. Ambos aspectos son cruciales a la hora de estimar la seguridad del sistema.

Por lo tanto, para cubrir las limitaciones encontradas en el estado-del-arte, en esta tesis se ha desarrollado un nuevo simulador con el que sea posible explorar diferentes alternativas de diseños de redes de sensores inalámbricas considerando distintos aspectos, como su arquitectura HW, su diseño SW, su tipo de despliegue o su seguridad. Como se puede observar en la Tabla 7, el simulador propuesto cumple con las especificaciones descritas al inicio de la sección. Además, existen pocos trabajos relacionados con simuladores de ataques como el descrito en este capítulo, como se verá más adelante.

En el ámbito de la seguridad, se han documentado un elevado número de ataques contra redes de sensores inalámbricas. Los análisis presentados en [51], [56], [57], [58], [59], [60] y [61] muestran hasta 20 tipos de ataques distintos específicos de redes de sensores inalámbricas. Uno de los principales problemas en el desarrollo de una técnica para simular ataques es como modelar de forma eficiente todos estos ataques. Para resolver este problema, en esta sección se propone un modelo de ataques que clasifica todos los ataques en únicamente cuatro categorías, dependiendo de sus efectos sobre la red. Cada categoría se modela mediante un atacante simple, lo cual facilitará la implementación en el simulador. La combinación de estos atacantes simples posibilita el modelado de la mayoría de los ataques reportados a redes de sensores.

Una vez que está disponible una plataforma que permite la simulación de redes inalámbricas con aspectos de seguridad, es posible utilizarla para diseñar software que evite ataques. Existen múltiples trabajos y técnicas en el área de contramedidas frente a ataques en redes de sensores inalámbricas, como por ejemplo las presentadas en [56], [62] y [63]. En [51] se presenta una clasificación de metas de seguridad. Están clasificadas como primarias y secundarias [53]. Las primarias se refieren a las metas de seguridad estándar como la confidencialidad, la integridad, la autenticación y la

disponibilidad Las secundarias son la validación de los datos, la auto-organización, la sincronización de tiempo y la localización segura. En [64], los autores analizan diferentes enfoques para la detección y mecanismos de defensa contra ataques a nivel de la capa de enlace. En [65] se describen distintos tipos de ataques, cuestiones relacionadas con la seguridad y sus contramedidas. También se presentan técnicas para impedir que los atacantes accedan al medio inalámbrico, como, por ejemplo, hibernar el nodo y usar comunicación mediante espectro ensanchado. Hay múltiples trabajos que presentan soluciones relacionadas con la encriptación de los datos para conseguir lograr mejorar la seguridad pero, como se comenta en [66] y en el próximo capítulo, es necesario medir la efectividad de estas soluciones en redes de sensores inalámbricas.

Existen pocos trabajos enfocados específicamente a la simulación de ataques en redes de sensores inalámbricas. En [67] y [68] se presenta un trabajo de simulación de ataques en redes de sensores inalámbricas llamada ASF (*Attack Simulation Framework*) basado en los simuladores OMNET++ y Castalia. Sin embargo, el problema de este trabajo es que no tiene en cuenta el software embebido en los nodos de la red, por lo que el tráfico de la red es generado basándose en funciones externas. Otra herramienta de simulación de ataques en las comunicaciones de la red es NETA [69]. NETA está construida basándose en OMNET++. Su principal problema radica en que únicamente permite la simulación de tres tipos diferentes de ataques. Existen modelos analíticos dirigidos a detectar y contrarrestar ataques como los presentados en [70], [71] y [72]. En [72] se describe un algoritmo de detección de distribución de gusanos (*Wormhole*) y se muestran los resultados de la simulación con el fin de demostrar sus bajas tasas de falsa tolerancia y falsa detección. En [73] se propone un esquema de detección de intrusión que detecta ataques por *Black Hole* y *Selective Forwarding*. En estos casos la simulación se ha utilizado para validar su eficiencia y su efectividad. En [74] se presenta una herramienta que simula ataques mediante la inyección de eventos. En [75] se utilizan diagramas de secuencia UML para describir y analizar posibles ataques en una red y en la capa de transporte.

3.3. Clasificación y modelado de ataques

Con el objetivo de permitir la evaluación de los efectos de los ataques más típicos que una red inalámbrica puede sufrir, se ha desarrollado un modelo que cubre un amplio número de tipos de ataques. El conjunto de alternativas seleccionadas se ha basado principalmente en las descritas en [64], [61] y en otros trabajos más específicos citados en el estado del arte de la sección anterior, así como en referencias específicas que se incluirán en la sección específica de cada tipo de ataque. De acuerdo con el estudio realizado, los ataques han sido clasificados en 5 categorías diferentes, dependiendo de sus efectos en los nodos y la red:

- Ataques que introducen ruido en la red
- Ataques que introducen paquetes en la red
- Ataques que introducen ruido y paquetes en la red
- Ataques que modifican el firmware/Hardware de un nodo
- Ataques que no afectan al comportamiento de la red

3.3.1. Ataques que introducen ruido en la red

El primer grupo de ataques, al que denominaremos “introductores de ruido”, incluye aquellos cuyo objetivo es la introducción de ruido o la reducción del tráfico en la red, con el objetivo de incrementar la probabilidad de perder paquetes. Utilizando distintas técnicas como puede ser la modificación del *routing* o la introducción de ruido en la red, estos ataques logran que haya paquetes que no lleguen a su destino. Por esto, estos ataques reducen el tráfico de la red, lo que tiene múltiples efectos en el rendimiento y el consumo del sistema. Las siguientes subsecciones muestran diferentes ataques incluidos en esta categoría.

3.3.1.1. Ataque por *Jamming*

Este ataque produce una denegación del servicio a usuarios autorizados, atascando el tráfico legítimo con una cantidad abrumadora de tráfico ilegítimo [76]. Se interrumpe la funcionalidad de la red al transmitir señales con mucha energía, que colisionan con los paquetes verdaderos. En definitiva, lo que busca el atacante es introducir ruido en algunos canales para romper la comunicación entre nodos. Hay muchas estrategias de ataque por Jamming, pero podemos clasificarlas según el modo de introducir el ruido en la red. De esta forma podemos hablar de Jamming por ruido, tonos, pulsos, barrido o seguimiento de frecuencias.

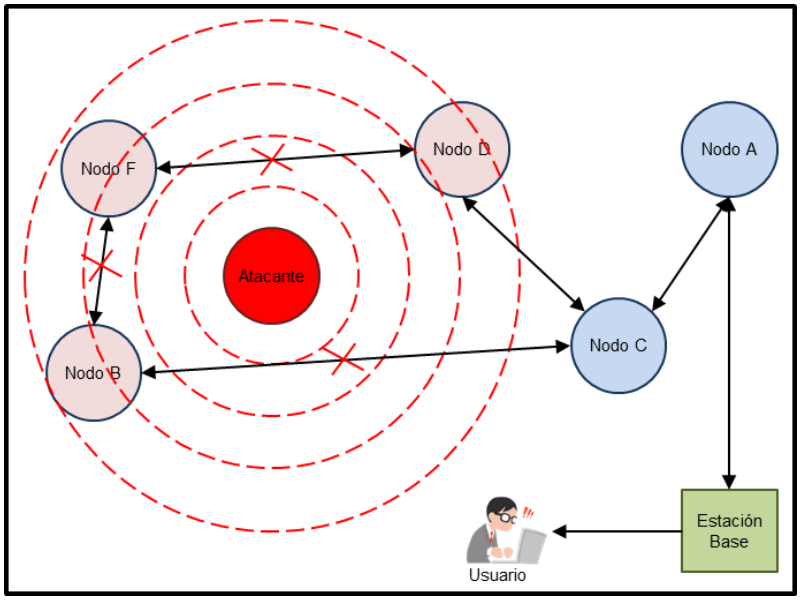


Figura 17: Ataque Jamming

Adicionalmente, los ataques por Jamming se pueden clasificar por el ancho de banda del ruido. De esta forma se puede hablar de ataques de:

- Banda-ancha o BBN (*Broad Band Noise*)
  - Genera ruido en todos los canales.
- Banda-parcial o PBN (*Partial Band Noise*)
  - Inserta ruido solo en una parte del espectro.
- Banda estrecha
  - Introduce ruido en un único canal.

La Figura 17 ilustra cómo se comporta el ataque por Jamming. El atacante produce ruido de manera intermitente o persistente produciendo interferencias en la comunicación. Como se puede ver en la figura, esta interferencia afecta directamente al menos a 3 nodos de la red (B, D y F).

3.3.1.2. Ataque *Collision*

En un ataque por colisión [77], el nodo atacante no sigue el protocolo de control de acceso al medio y produce colisiones con las transmisiones del nodo vecino mediante el envío de un paquete corto ruidoso. Los paquetes colisionan cuando dos nodos intentan transmitir mensajes en la misma frecuencia al mismo tiempo, produciendo la corrupción de los paquetes. Este ataque puede causar una gran cantidad de interrupciones en el funcionamiento de la red.

3.3.1.3. Ataque *Resource Exhaustion*

Este ataque consiste en repetir colisiones y múltiples retransmisiones hasta que el nodo atacado se queda sin energía o sin recursos de cómputo para procesar la información [78]. El nodo malicioso transmite o solicita paquetes continuamente a través del canal de comunicación. Aunque este ataque normalmente inserta paquetes falsos en la red, en realidad su efecto final es el de reducción de paquetes en la red, ya que su efecto principal es la colisión de estos paquetes falsos con los paquetes verdaderos reduciendo el tráfico efectivo de la red. Debido a esto, este ataque se agrupa con los ataques que introducen ruido en la red en vez de los que introducen paquetes (recordemos que su agrupación es por efectos de los ataques, no por sus estrategias).

3.3.1.4. Ataque *DoS*

Los ataques de denegación de servicios [79] (DoS*, Denial of service*) es un ataque a una red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos.

Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la víctima. Estos pueden causar mucho daño a sistemas alimentados por batería. En este trabajo se presta especial atención a un tipo de ataque DoS llamado *Path-Based DoS*. En un ataque DoS, se envía al nodo atacado una gran cantidad de paquetes, haciendo que el nodo se sobrecargue y no pueda seguir prestando servicios; por eso se le denomina "denegación", pues hace que el servidor no dé abasto a la cantidad de solicitudes. Como se puede ver en [79], aunque la mayoría de los métodos para efectuar este ataque consistan en la introducción de muchos paquetes con destino el nodo atacado, su mayor efecto es que el nodo atacado no puede seguir prestando servicios.

3.3.1.5. Ataque *Homing*

En un ataque por *Homing* [80], el atacante estudia el tráfico de la red para deducir la localización de los nodos críticos, tales como nodos coordinadores (*router*) o los vecinos de la estación base. El ataque consiste en deshabilitar esos nodos. Este ataque puede tener similitudes con el ataque por agujero negro.

3.3.1.6. Ataque *Black Hole*

El ataque por agujero negro [81] consiste básicamente en alterar el *routing* de la red con el objetivo de atraer todos los paquetes hacia el nodo atacante y, sigilosamente, éste los vaya descartando o retirándolos del tráfico de la red. En la Figura 18 se muestra un ataque con esta estrategia, donde todos los paquetes enviados por los nodos I, G, D E y D son capturados por el atacante y desechados.

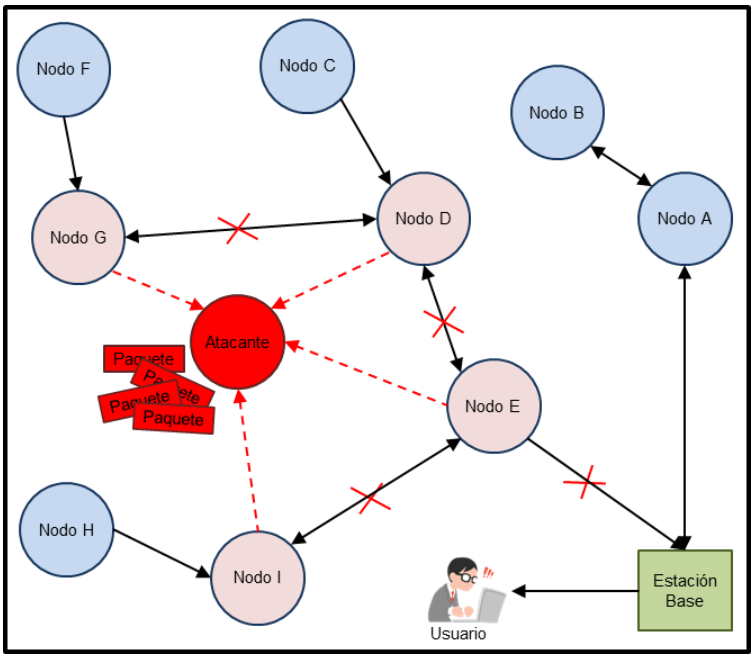


Figura 18: Ataque Black Hole

3.3.1.7. Ataque *Selective Forwards*

En las redes multi-salto, los nodos asumen que cuando envían un mensaje a un nodo con el que no tienen comunicación directa, éste llega correctamente. Para enviar mensajes a dicho nodo no accesible directamente, el emisor tiene que enviar el paquete a nodos accesibles para que éstos se lo reenvíen al destinatario final mediante una serie de "saltos" (transmisiones) entre nodos directamente accesibles. Sin embargo, un nodo malicioso podría ser el encargado de recibir los mensajes y simplemente desechar o no propagar los que él crea conveniente.

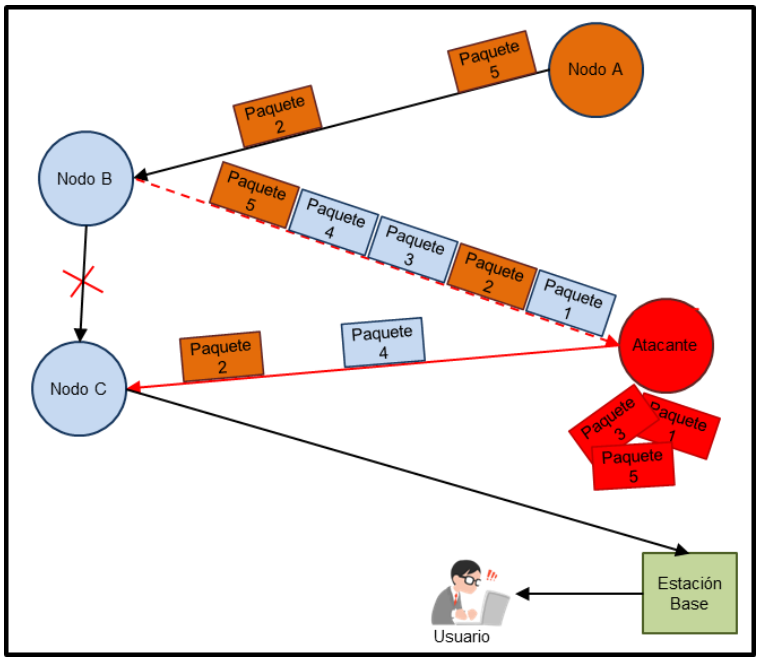


Figura 19: Selective Forward 1

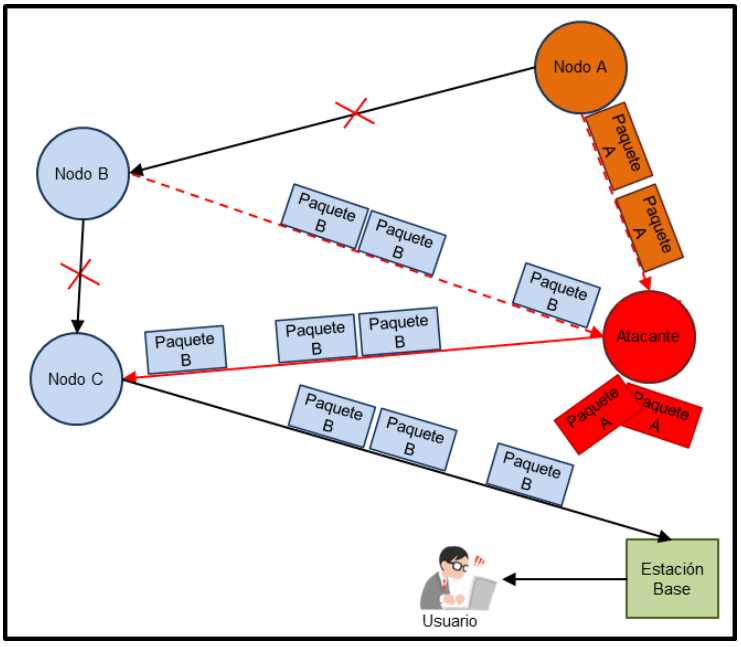


Figura 20: Selective Forward 2

El procedimiento que se sigue en este tipo de ataques es muy similar al ataque descrito en la Sección 3.3.1.6 (Ataque por agujero negro). Primero, un nodo atacante convence a la red de que él es el nodo más cercano a la estación base, atrayendo todos los paquetes a él mismo. Una vez que los nodos de la red confían en él y los mensajes

pasan por el nodo malicioso, éste solo tendrá que desechar todos los paquetes (Figura 18) y eliminar un número aleatorio de paquetes (Figura 19) o suprimir unos paquetes específicos (Figura 20).

3.3.2. Ataques que introducen paquetes en la red

El segundo grupo de ataques, a los que llamaremos “inyectores de paquetes falsos”, engloban aquellos cuyo principal efecto es un aumento del tráfico en la red. Normalmente, el objetivo de estos ataques es que los nodos de la red reciban más paquetes de lo habitual, buscando que la red se congestione o que el tiempo de procesado en los nodos aumente. La consecuencia de estos ataques puede ser un aumento del tiempo de ejecución y una reducción del ciclo de vida de un nodo o de la red. Hay que tener en cuenta que las redes de sensores inalámbricas pueden estar desplegadas con diferentes topologías, pudiendo ser necesario que los mensajes de un nodo pasen siempre por un nodo vecino. En este caso, si el funcionamiento de un nodo es erróneo o la batería se agota, se podrá quedar aislado de la red el conjunto de nodos que envían sus mensajes a través el nodo atacado. Los ataques que aumentan el tráfico considerados en este trabajo se describen en las siguientes subsecciones.

3.3.2.1. Ataque *Interrogation*

El ataque por interrogación [80] explota el mecanismo de confirmación (*handshake*) que muchos protocolos usan para evitar problemas de pérdida de integridad de los mensajes en la red. El ataque por interrogación consiste en enviar mensajes RTS (*Require To Send*) para obtener respuestas CTS (*Clear to Send*) de un nodo vecino.

3.3.2.2. Ataque *Energy Drain*

El objetivo de estos ataques es incrementar el consumo de energía para agotar la batería del nodo, cuya recarga o reemplazo suele ser difícil, cara y/o complicada. Para ello los atacantes pueden comprometer los nodos inyectando mensajes en la red o generando una gran cantidad de tráfico. Estos falsos mensajes pueden causar falsos requisitos en los nodos de la red, a los cuales responden consumiendo energía de sus baterías. Un objetivo de este ataque [82] podría ser eliminar nodos de la red, degradando el rendimiento y finalmente dividiendo la red en partes. Esto permite al atacante tomar el control de una parte de la red, insertando un nuevo nodo atacante como enlace.

La Figura 21 muestra un nodo atacante generando mensajes falsos continuamente. Sus nodos vecinos (C y G) responden al ataque y finalmente se quedan sin batería.

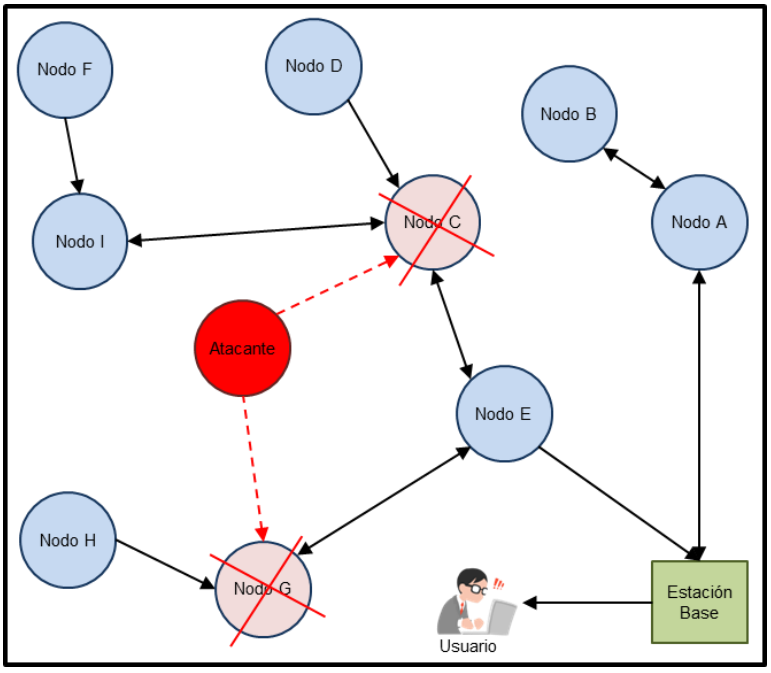


Figura 21: Ataque *Energy Drain*

3.3.2.3. Ataque *Hello Flood*

En este caso (ataque *Hello Flood* [83]), el atacante típicamente intenta consumir la energía de los nodos atacados mediante mensajes específicos. Un atacante con un alto poder de transmisión puede enviar paquetes "HELLO" (usado en múltiples protocolos para descubrir nodos nuevos) para convencer a los nodos de la red de que un nuevo nodo pide conexión, provocando que muchos nodos le contesten y gasten su energía enviando paquetes a este nodo imaginario.

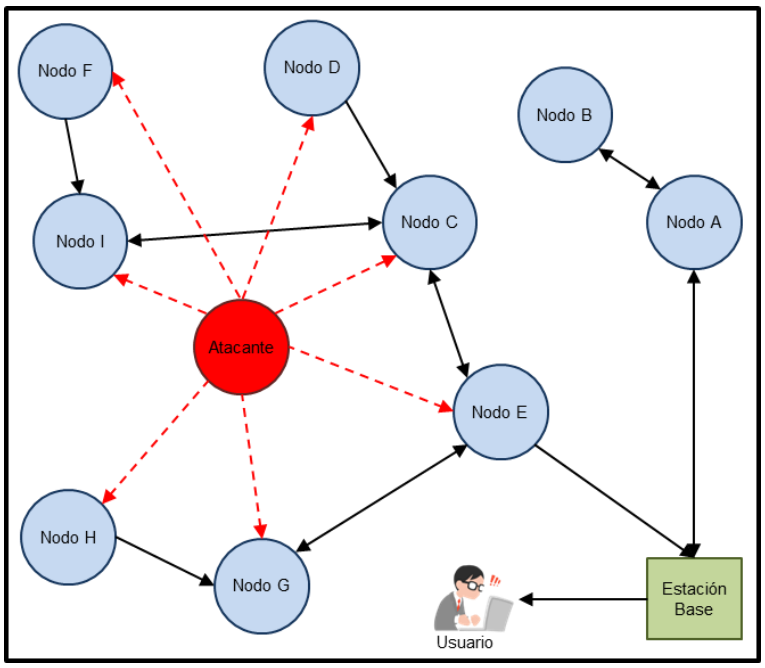


Figura 22: Ataque *Hello Flood*

En la Figura 22 se muestra un ejemplo de este ataque, donde el nodo atacante envía a varios nodos paquetes "*Hello*" (en modo *broadcast*), para convencer a la red de que él es un nodo vecino genuino, provocando su respuesta y consumo de energía.

3.3.2.4. Ataque *Misdirection*

Este ataque [84] consiste en la introducción de mensajes sin destino conocido en la red o con destino inalcanzable. El atacante introduce el paquete con un destino que no

existe en la red. El objetivo es que el paquete intruso viaje sin dirección por la red. Esto aumenta el consumo de los nodos al tener más paquetes que procesar, al tiempo que aumenta la latencia de los paquetes reales, haciendo que solo unos pocos alcancen la estación base.

3.3.2.5. Ataque *Flooding*

El atacante que implemente un ataque *Flooding* [85] mandará continuamente solicitudes de nueva conexión, hasta que los recursos del nodo al que le solicitan la conexión queden exhaustos. Esto puede producir serios daños al nodo atacado, al reducir la vida de su batería.

3.3.2.6. Ataque *Sink Hole*

En el ataque por *Sink Hole* [86], el adversario intenta atraer todo el tráfico cercano a un cierto área, a través de un nodo comprometido. Ese nodo comprometido se coloca en el centro del área que se desea atacar y crea una “esfera de influencia”, atrayendo todo el tráfico de los nodos destinado a una estación base.

3.3.3. Ataques que introducen ruido y paquetes en la red

La tercera categoría agrupa a los ataques que causan efectos más complejos en la red, mezclando la inserción de ruido y paquetes comentada en anteriores secciones. Estos ataques causan la pérdida de ciertos paquetes pero, simultáneamente, introducen nuevos paquetes en la red. Además, pueden modificar el contenido o el tipo de los paquetes transmitidos, causando una reducción de ciertos tipos de paquetes y, al mismo tiempo, aumentando la frecuencia de otros. Esto puede tener distintas consecuencias en

la red, que serán analizados en las siguientes subsecciones que muestran los ataques seleccionados.

3.3.3.1. Ataque *Spoofing*

En un ataque por suplantación [87], un atacante asume la función de un nodo de la red falsificando sus datos y consiguiendo una ventaja ilegítima. Este ataque normalmente busca la alteración de las rutas (*routing*) de la red para debilitar al sistema. Existen tres técnicas básicas de ataque por suplantación, con distintos objetivos a la hora de llevar a cabo este tipo de ataque:

- Lazo en la red:
  - Consiste en modificar el *routing* de la red, para que un paquete realice un bucle infinito y no llegue nunca a su destino.

- Atraer el tráfico:
  - Mediante esta técnica conseguimos que el tráfico pase por el atacante y éste lo modifique a su antojo.

- Repeler el tráfico:
  - Se consigue sobrecargar una parte de la red y disminuir el uso en otra.

3.3.3.2. Ataque *Sybil*

El ataque por *Sybil* [88] es una modificación de los caminos (*routing*) de la red buscando atraer el tráfico hacia los nodos atacantes, con el objetivo de aislar de la red ciertos nodos. Cuando estos nodos no se pueden comunicar más, el atacante procede a enviar paquetes falsos con el objetivo de suplantar la identidad de los nodos aislados o incomunicados. Como se puede apreciar en la Figura 23, el atacante (nodo rojo) consigue incomunicar al nodo H y lo suplanta para poder atraer su tráfico. Gracias a esto, el atacante tiene el control para poder modificar los paquetes dirigidos a otros nodos a su antojo.

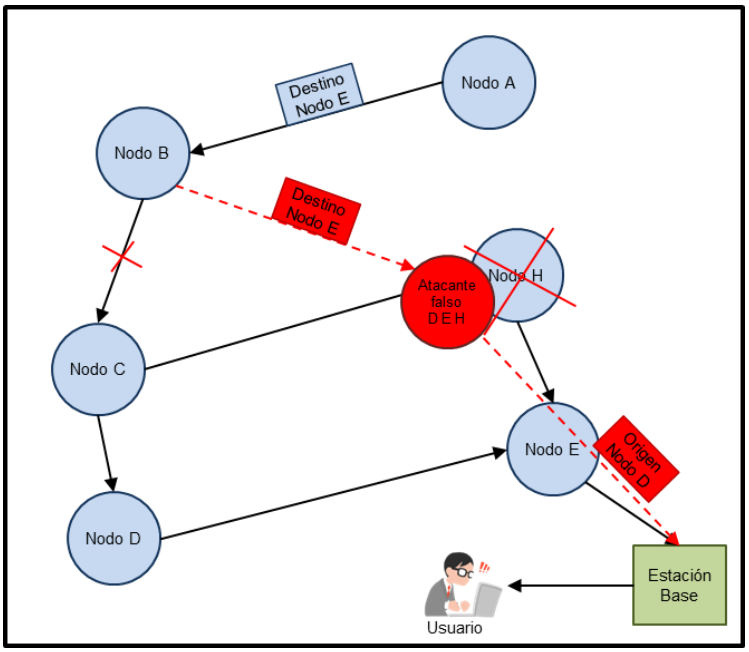


Figura 23: Ataque *Sybil*

3.3.3.3. Ataque *Replication*

En el ataque *Replication* o por replicación [89] [90] [91], se busca introducir diferentes nodos en la red que suplanten la identidad de un nodo legítimo. Aunque el ataque por replicación pueda parecerse al ataque *Sybil* descrito en la Sección anterior, éstos difieren en una característica clave. En el ataque por *Sybil*, es un único atacante el que se hace pasar por múltiples identidades, mientras que en el ataque por replicación son múltiples nodos los que se hacen pasar por un único nodo de la red. Por lo tanto, el ataque *Sybil* es más fácil llevarlo a cabo ya que únicamente es necesario un atacante para tener éxito. En cambio, el ataque por replicación requiere insertar múltiples atacantes para tener éxito. Debido al mayor número de atacantes, las posibilidades de detectar este ataque aumentan. La Figura 24 muestra un ejemplo de ataque por replicación.

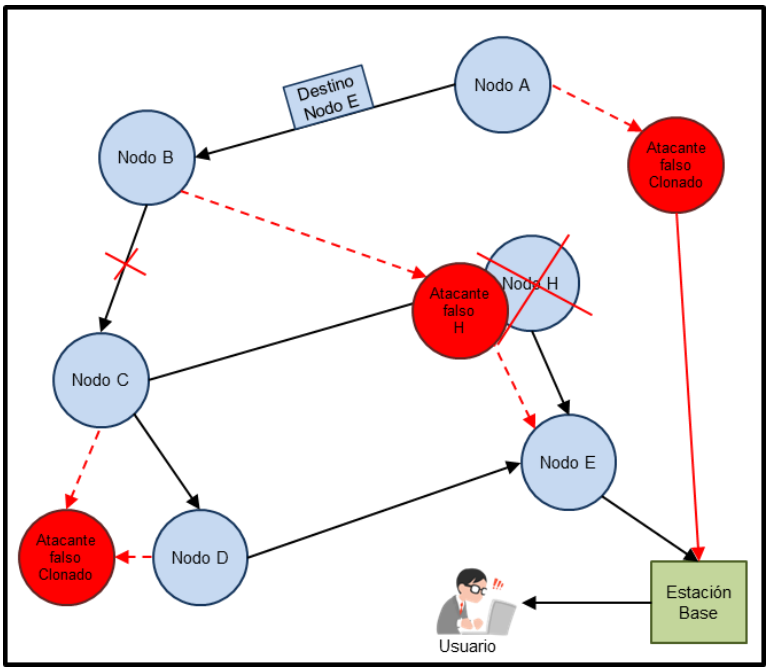


Figura 24: Ataque *Node Replication*

3.3.3.4. Ataque *Looping in the network*

Este ataque [92] busca modificar las rutas (*routing*) de la red buscando afectar la transmisión de datos. Como se puede observar en la Figura 25, el atacante (Nodo rojo) informa a otro nodo que es el nodo final de la cadena y que es el nodo que más cerca tiene la estación base (a únicamente un salto). Con esto consigue recibir todos los paquetes del nodo C para volver a inyectarlos en la red de forma que se genere un bucle infinito que colapse el sistema.

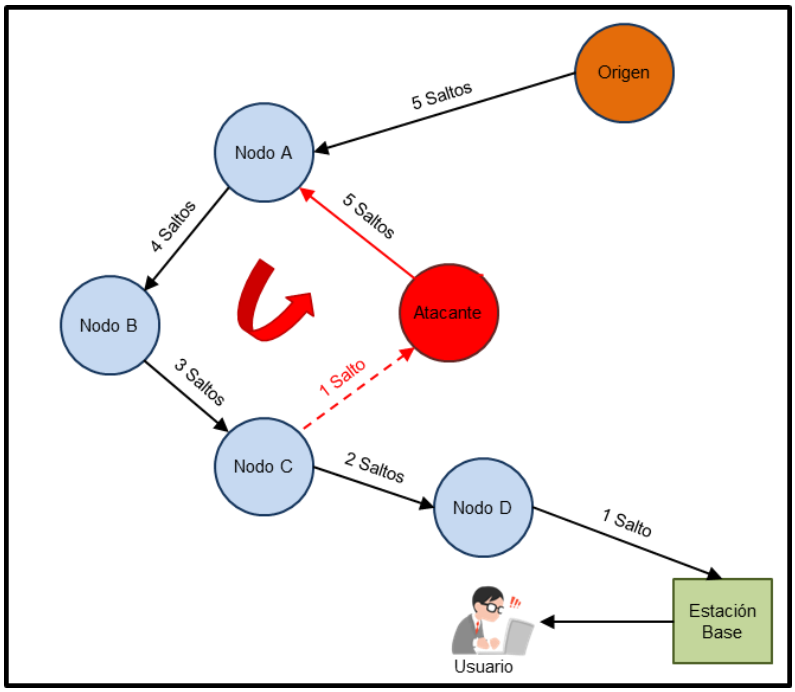


Figura 25: Ataque *Looping in network*

3.3.4. Ataques que modifican el Firmware/Hardware de un nodo

Esta categoría agrupa a los ataques que no tienen una interacción directa con el tráfico de red, sino que alteran el funcionamiento del nodo físico atacado. Estos ataques modifican directamente el comportamiento de un nodo para que su funcionalidad sea maligna. A continúan se describen los ataques seleccionados e integrados en esta categoría.

3.3.4.1. Ataque *Application*

Este ataque implica una modificación del firmware o software embebido en el nodo de red. El atacante normalmente necesita tener acceso directo al nodo físico y

capacidad para modificar su programación con el fin de introducir código que produzca un funcionamiento maligno.

3.3.4.2. Ataque *Overwhelm*

Es un ataque directo al sensor del nodo, para que el envío de paquetes aumente o mande información errónea y, en consecuencia, el consumo del nodo o de la red aumente. El atacante normalmente necesita acceso directo al nodo para provocar estímulos en el sensor o lecturas erróneas.

3.3.5. Ataques que no afectan el comportamiento de la red

Este último grupo de ataques no está incluido en el simulador ya que no afectan al comportamiento de la red ni de los nodos. El objetivo de dichos ataques es obtener información de forma ilegítima (robar datos) sin interactuar ni modificar el comportamiento de la red ni de los nodos. Por lo tanto, se trata de ataques pasivos, que roban datos sin ser detectados. Aun siendo ataques peligrosos por el robo de información, estos no afectan en ningún sentido a la red: no modifican ni el tráfico, ni aumentan el consumo de los nodos, sino que únicamente escuchan o acceden a información confidencial. Los ataques agrupados en esta categoría son los presentados en las siguientes subsecciones.

3.3.5.1. Ataque *Sniffing*

Los ataques por *Sniffing* capturan paquetes de la red. Si los paquetes de la red no están encriptados, los datos de los mensajes pueden ser capturados y leídos fácilmente por un ataque de este tipo. *Sniffing* se refiere al proceso usado por el atacante para capturar el tráfico de la red usando un *sniffer* (programa y hardware específicos para

capturar mensajes en redes). Cuando un paquete es capturado por un *sniffer*, su contenido puede ser analizado. Los *sniffers* pueden ser utilizados por los atacantes para capturar datos importantes como contraseñas, configuración de la red/nodo, etc.

3.3.5.2. Ataque *Tampering*

Los ataques por *Tampering* [93] requieren acceso directo al hardware físico del nodo para robar sus datos internos. A través de estos ataques es posible obtener información sensible, como por ejemplo llaves criptográficas.

3.4. Modelo de atacantes

Como se ha descrito en la sección anterior, después de haber realizado un estudio de los ataques típicos a las redes de sensores inalámbricas, estos se han clasificado según sus efectos en 5 categorías. En esta sección, dichas categorías se van a representar mediante tres diferentes modelos de atacantes. Dichos modelos serán posteriormente introducidos en el simulador para poder cubrir todos los ataques presentados en la sección 3.3. Como se mencionó anteriormente, los efectos de los ataques se han dividido en 5 grupos distintos: 4 grupos con efectos en la red/nodo y 1 grupo de ataques “sin efecto” directo. Los tres modelos de atacantes que se presentarán a continuación permiten, de manera individual o conjunta, modelar las 4 categorías que producen efectos y, por ello, simular todos los ataques estudiados en la sección previa. Los modelos de atacantes propuestos son:

- Atacante que inyecta paquetes en la red.
- Atacante que reducen el tráfico de la red.
- Atacante directo.

3.4.1. Atacante que reduce el tráfico de la red

Como se ha mencionado anteriormente, un efecto típico de muchos ataques es la reducción del tráfico de la red. Esta reducción puede ser resultado de distintas estrategias de ataque. Por ejemplo, como se mostró en la sección 3.3.1.1, el ataque por Jamming corrompe la funcionalidad de la red emitiendo señales de alta energía lo que provoca que los paquetes enviados por los nodos legítimos no puedan ser recibidos por los nodos de la red.

Todos estos ataques, que conllevan una pérdida de paquetes, serán modelados mediante un atacante "introductor de ruido" con distintas configuraciones. Un ejemplo gráfico puede observarse en la Figura 26, donde el atacante introduce ruido en la red que corrompe las conexiones del nodo 4, pudiendo llegar a aislarlo. Para simular los distintos ataques que conllevan una reducción del tráfico en la red, el atacante podrá ser configurado con distintos parámetros:

- ***Links*:** Lista de los enlaces de comunicación o pares de nodos que son afectados por el nodo atacante.

- ***Power***: Cantidad de ruido que será inyectada en cada enlace definido en el parámetro "Links". El ruido se modela como un porcentaje adicional de probabilidad de pérdida de paquete que se añadirá a la probabilidad original del enlace afectado.

- ***NumPackets*:** Porcentaje de paquetes que son afectados por el incremento de la probabilidad de pérdida de mensajes. Por lo tanto solo el porcentaje "NumPackets" de paquetes verá afectada su probabilidad de pérdida de mensajes, no siendo el resto afectados por el nodo atacante.

- ***Time*:** Define los rangos de tiempo en los cuales el atacante afecta a la red. El agresor puede estar activo o inactivo múltiples veces durante la simulación, modelándose sus intervalos de actividad mediante este parámetro.

- ***TypePackets***: El ataque puede afectar a un determinado tipo de paquetes. Este parámetro permite la simulación de ataques selectivos, que solo afectan a un tipo de ataques.

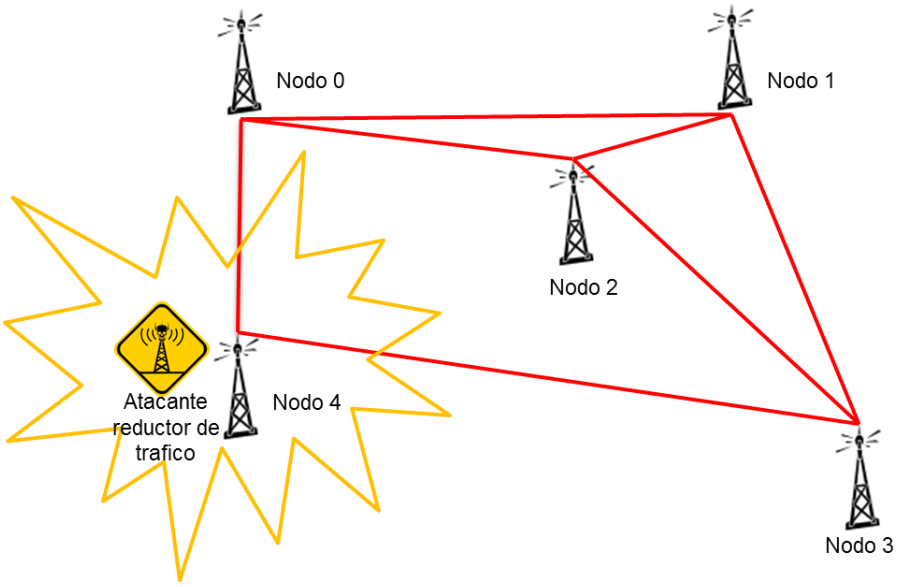


Figura 26: Atacante introductor de ruido

En resumen, cualquier ataque que tiene como efecto una pérdida de paquetes en la red podrá ser modelado mediante uno o varios “atacantes introductores de ruido”, cada uno de ellos con unos parámetros de configuración específicos.

3.4.2. Atacante que inyecta paquetes en la red

Otro efecto típico de muchos ataques es el aumento del tráfico en la red. Este aumento es debido a que el atacante manda paquetes falsos con diversos objetivos (Figura 27). Por ejemplo, en un ataque por interrogación, el atacante envía repetidamente paquetes RTS (*Require To Send*), o en un ataque por *Hello Flood* el atacante intenta descargar la batería del nodo atacado mandándole continuamente paquetes “HELLO”. Como se puede observar, estos ataques guardan ciertas similitudes ya que ambos inyectan en la red diferentes tipos de paquetes. Por lo tanto, estos ataques podrán ser modelados con el mismo atacante: un “atacante inyector de

paquetes falsos". Este nodo será el responsable de introducir paquetes falsos en la red. Dependiendo del ataque que se quiera simular y analizar, el atacante podrá inyectar diferentes tipos de paquetes ("HELLO", "RTS", DATOS, ALEATORIO, etc.) en la red. La estructura del paquete a inyectar podrá ser tanto predefinida como definida por el usuario. Una vez que han sido introducidos en la red, estos paquetes podrán ser recibidos por los nodos legítimos de la red ya que formalmente tienen una estructura correcta. Para simular todos los ataques estudiados, el atacante tendrá que ser configurado con distintos parámetros para poderse amoldar a cada situación. Los parámetros de configuración que se pueden definir en el atacante son los siguientes:

- ***Frequency*:** Define la tasa de paquetes falsos inyectados en la red.
- ***TypePackets***: Define el tipo de paquetes que se van a inyectar. Pueden ser implementados distintos tipos de paquetes, para abarcar todos los ataques descritos en secciones anteriores: desde paquetes "HELLO", RTS o CTS a paquetes de tipo aleatorio.
- ***Time***: Define los rangos de tiempo en los cuales el atacante está activo. El atacante puede activarse y apagarse las veces que sean necesarias durante la simulación.
- ***Nodes Destine*:** Lista que especifica los nodos destino a los que se les inyecta paquetes.
- ***Broadcast*:** Si este atributo está definido, cada paquete inyectado será enviado en modo *broadcast* y llegará a todos los nodos visibles de la red.

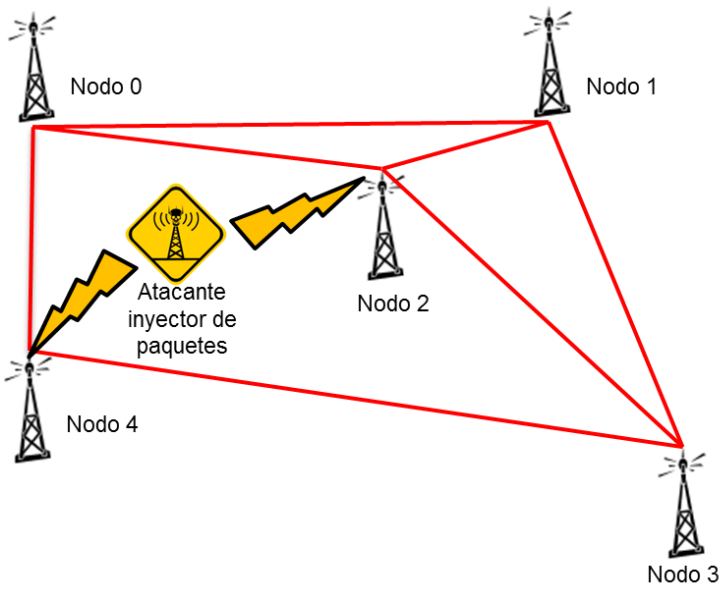


Figura 27: Atacante inyector de paquetes falsos

En resumen, todos los ataques que tiene como efecto una inyección de nuevos paquetes en la red podrán ser modelados utilizando el "atacante inyector de paquetes falsos" con unos parámetros de configuración específicos.

3.4.3. Atacante directo

Cuando se compara la clasificación realizada en la sección 3.3 de los tipos de ataques con los efectos que producen los 2 tipos de atacantes anteriormente propuestos, se observa que no todos los ataques pueden ser modelados con dichos atacantes. Por ello, será necesario definir un tercer atacante que modele ataques "físicos" no cubiertos por otros atacantes. Este tercer atacante se denominará "atacante directo". Los ataques no cubiertos con los dos modelos de atacantes anteriormente descritos, como se vio en la Sección 3.3.4, son los ataques de aplicación y el de *Overwhelm*. Estos ataques no interactúan directamente sobre la red, sino sobre la arquitectura HW/SW del nodo. Por ejemplo, en el ataque por aplicación el agresor modifica directamente el firmware que un nodo de la red. Para modelar estos ataques, se propone utilizar el "atacante directo", el cual será el encargado de cargar el nuevo firmware en el nodo durante la simulación, permitiendo de esta forma modificar el

comportamiento del nodo atacado. Por lo tanto, se trata de un modelo que permite modificar totalmente el software que se ejecuta en un nodo, sustituyendo el software legítimo por comportamiento malicioso. Este atacante es configurado mediante dos parámetros:

- ***Application***: Define la localización de la nueva aplicación que será cargada en el nodo atacado y ejecutada tras el ataque.
- ***Time***: Define el instante de tiempo en el cual la modificación del firmware es cargada.

3.5. Relación entre ataques y atacantes

Una vez se han clasificado los ataques típicos en grupos dependiendo de sus efectos (Sección 3.3) y se han propuesto modelos de los atacantes que permiten emular dichos efectos (Sección 3.4), únicamente resta describir cómo utilizar los atacantes propuestos para poder modelar cada ataque típico a una WSN. En la sección anterior se han propuesto tres modelos de atacantes:

- **Atacante reductor de tráfico:** Introduce ruido en la red para reducir el tráfico.
- **Atacante inyector de paquetes falsos:** Inyecta nuevos paquetes en la red.
- **Atacante Directo:** Modela ataques que modifican directamente el software del nodo.

La siguiente tabla (Tabla 4) muestra el atacante (o combinación de atacantes) que permite modelar cada uno de los tipos de ataques típicos anteriormente identificados. La Tabla 4 agrupa los ataques en cinco grupos. En el primer grupo se incluyen los ataques que son modelados mediante un “atacante reductor del tráfico”. En el segundo grupo se integran los ataques modelados con un “atacante inyector de paquetes falsos”. El tercer grupo incluye ataques más complejos, como el ataque por replicación de nodo o el ataque *Sybil*, que se modelarán combinando dos atacantes: un “atacante reductor de tráfico” conjuntamente con un “atacante inyector de paquetes falsos”. El cuarto grupo será el formado por los ataques que modifican el Hardware o el Software de un nodo,

cambiando su comportamiento. Dicho grupo de ataques será modelado con un "atacante directo". El último grupo presentado en la Tabla 4 incluye los ataques que no afectan directamente al comportamiento de la red. Estos ataques no se modelarán en el simulador ya que no afectan al comportamiento del sistema por tratarse de ataques pasivos, es decir que roban información sin alterar a la red y nodos.

Tabla 8: Relación entre ataques y atacantes

Atacante utilizado	Ataque
Atacante reductor de tráfico	*Jamming*
	Collision
	Resource Exhaustion
	DoS attacks
	Homing attack
	Black Hole attack
	Selective Forwarding
Atacante inyector de paquetes falsos	*Interrogation*
	Energy Drain
	Hello Flood
	Misdirection
	Flooding
	Sink Hole
Introductor de ruido + Inyector de paquetes falsos	*Spoofing*
	Sybil
	Node replication
	Looping in the network
Atacante Directo	*Application*
	Overwhelm
Sin efectos sobre la simulación	*Sniffing*
	Tampering

3.6. Implementación de los ataques en el simulador

En la Sección 2.3 se ha presentado la infraestructura desarrollada para simular y analizar las prestaciones de redes de sensores inalámbricas. En esta sección se va a extender dicho entorno para soportar la simulación de ataques. Dichos ataques pueden ser modelados mediante una combinación de los atacantes descritos en la sección anterior. Por esta razón, en esta sección se describe cómo los atacantes presentados anteriormente son soportados por el simulador descrito en la Sección 2.3, con objeto de poder simular ataques.

3.6.1. Ataques que reduce el tráfico de la red

Como muestra la Tabla 4, el atacante que reduce el tráfico en la red permite modelar distintos riesgos de seguridad, como los ataques por Jamming, colisión, denegación de servicio o *Sybil*. Para ello, solo es necesario definir el atacante con los parámetros de configuración adecuados. La definición de un "atacante reductor del tráfico de la red" en el simulador se realiza utilizando una llamada a función que responde a la estructura presentada en la Figura 28. Los argumentos de dicha función son los parámetros de configuración presentados en la Sección 3.4.1.

```
LinkNoise (links[], power[],numPackets, time[], typePackets)
```

Figura 28: Definición del atacante reductor de tráfico

Dichos parámetros definen al atacante y determinan el ataque cuyo efecto se está modelando. Más aún, dichos parámetros no solo permiten simular distintos ataques, sino que también permiten definir distintas estrategias de ataque. Por ejemplo, en el ataque por Jamming se han identificado distintas estrategias de ataque, como el Jamming por Banda-parcial o PBN. Para modelar esta estrategia tan solo es necesario especificar que el porcentaje de paquetes afectados por el atacante es del 50%. Un

ejemplo de cómo se usa un “atacante que reduce el tráfico en la red” para modelar un ataque por Jamming se muestra en la Figura 29.

```
Link link1 = {1,6};              //Jamming entre el nodo 1 y el 6
Link link2 = {1,2};              //Jamming entre el nodo 1 y el 2
Link links[2] = {link1,link2};
int power = 50; //Reducción en % de la probabilidad de recepción
int numPackets = 100;            //% de paquetes afectados
long time[2] = {100,100};        //Intervalo de tiempo
int packetType = ALL;            //Todos los paquetes afectados
LinkNoise( links, power, numPackets, time, packetType);
```

Figura 29: Definición de un ataque Jamming con un atacante reductor del tráfico

Con esta configuración, el atacante reduce un 50% la efectividad de los canales de comunicación entre los nodos 1-6 (link1) y 1-2 (link2). Este ataque no es selectivo (no se limita a un tipo específico de paquetes), sino que la reducción afecta a todos los paquetes que transiten por dichos canales de comunicación.

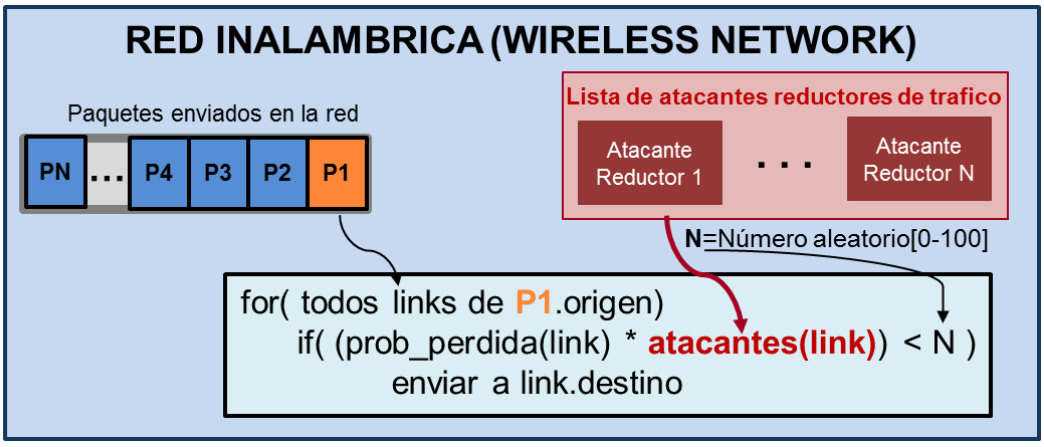


Figura 30: Simulación con atacantes introductores de ruido

Como muestra la Figura 26, el nodo atacante reductor de paquetes es el responsable de la pérdida de la calidad del canal en los enlaces especificados. Para permitir la simulación eficiente de estos nodos atacantes, se ha modificado el modelo de red

presentado en la sección 2.3.3 y mostrado en la Figura 11. Básicamente, el atacante modifica la probabilidad de pérdida de paquete para un cierto número de mensajes durante un periodo de tiempo predefinido. En la Figura 30 se puede ver el nuevo modelo de red, en el que se incluye esta nueva probabilidad generada por el atacante propuesto. Cuando un paquete tiene que ser transferido al nodo receptor, la probabilidad de recepción incluye la probabilidad del enlace original ("prob_perdida" en la Figura 30) más el ruido adicional introducido por el atacante ("atacantes" en la Figura 30).

3.6.2. Ataques que inyectan paquetes en la red

Este tipo de nodo atacante introduce paquetes falsos en la red con diferentes motivaciones. Los falsos mensajes son procesados por nodos legítimos porque el nodo atacante forma un paquete con una estructura formalmente correcta. En la Figura 18 se muestra la definición de este tipo de atacantes con sus distintos parámetros. La estructura del nuevo paquete inyectado en la red es definida por los parámetros con los que se especifica el atacante. Dichos parámetros permiten definir desde la frecuencia de inyección de paquetes hasta su tipo, modo de ataque, etc.

```
FakeInjec(frequency,typePacket,time[],destines[],broadcast)
```

Figura 31: Definición del atacante inyector

Un ejemplo de declaración de un ataque que se modela con un atacante inyector de paquetes falsos se presenta en la Figura 32.

```
int frequency = 5;                  //Un mensaje cada 5 segundos
int typePacket = CTS;               //Tipo de paquete CTS
long time[1] = {100};               //Intervalo de tiempo
int destines[1] = {2};              //Destino: nodo 2
int broadcast = FALSE;              //No es en broadcast
FakeInjec(frequency, typePacket, time, destines, broadcast);
```

Figura 32: Definición de un ataque usando un atacante inyector

Dicha declaración modela un ataque por interrogación, en el cual el agresor envía continuamente paquetes de tipo RTS/CTS (*Request to Send/Clear to Send*). En el código del ejemplo anterior, el nodo atacante inyecta en la red paquetes de tipo CTS cada 5 segundos, con destino al nodo 2.

Para la simulación de este atacante, es necesario que el nodo agresor incluya paquetes en la cola de transmisión del modelo de la red. Una vez que el paquete se ha introducido en la cola, es el modelo de red descrito en la Sección 2.3.3 el encargado de transmitir este falso paquete, ya que lo considera como un paquete genuino. En la Figura 33 se puede ver una ampliación de la Figura 11, con la introducción de este nuevo atacante en el modelo de la red.

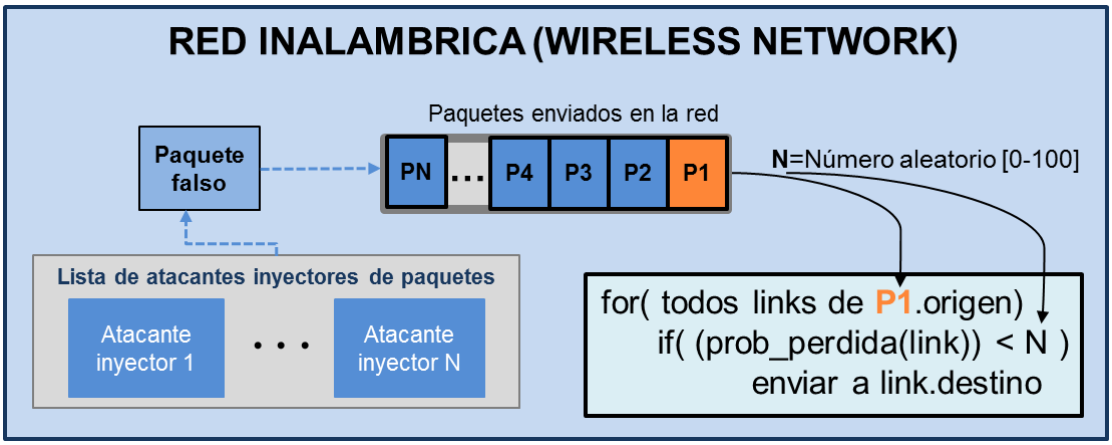


Figura 33: Simulación con atacantes inyectores de paquetes falsos

3.6.3. Ataques que introducen ruido y paquetes en la red

La simulación de ataques que requieren reducir el tráfico en la red al tiempo que introducen nuevos paquetes falsos se implementa en el simulador mediante la actuación simultánea de los dos atacantes anteriormente descritos. Por ejemplo, un ataque por *Spoofing* altera las rutas (*routing*) de la red con el objetivo de que el nodo atacante se haga pasar por un nodo legítimo, lo que le permite falsificar mensajes y así obtener una ventaja ilegítima. Para implementar este ataque en el simulador es necesario usar un

"atacante reductor del tráfico" junto con un "atacante inyector de paquetes falsos". El atacante reductor de tráfico en la red será el responsable de aislar al nodo atacado, debilitando o inutilizando los enlaces de dicho nodo para que no reciba ningún paquete (modificación de las rutas de la red). De forma simultánea, el atacante inyector de paquetes falsos se encarga de generar nuevos paquetes, con el objetivo de sobrecargar una parte de la red. Un ejemplo en el que modela un ataque por *Spoofing* se muestra en la Figura 34.

```
Link link1 = {3,5};       //Rompe el enlace entre los nodos 3 y 5
Link link2 = {1,5};       //Rompe el enlace entre los nodos 1 y 5
Link links[2] = {link1,link2};
int power = 50; //Reducción en % de la probabilidad de recepción
int numPackets = 100;             //% de paquetes afectados
long time[2] = {100,100};         //Intervalo de tiempo
int packetType = ALL;             //Todos los paquetes afectados
LinkNoise( links, power, numPackets, time, packetType);


int frequency = 1;                //Un mensaje cada segundo
typePacket = APP;                 //Tipo de paquete: APP
long time[1] = {100};             //Intervalo de tiempo
int nodesDestine[2] = {3,4};      //Destinos: nodo 3 y 5
int broadcast = FALSE;            //No es en broadcast
FakeInjec(frequency, typePacket,time,nodesDestine, broadcast);
```

Figura 34: Definición de ataque con dos atacantes: inyector y reductor del tráfico

Mediante esta implementación se consigue aislar completamente al nodo 5, al tiempo que se inyectan paquetes para sobrecargar la sub-red formada por los nodos 3 y 4. La Figura 35 muestra como ambos atacantes trabajan conjuntamente, a nivel modelo de red, para simular ataques complejos.

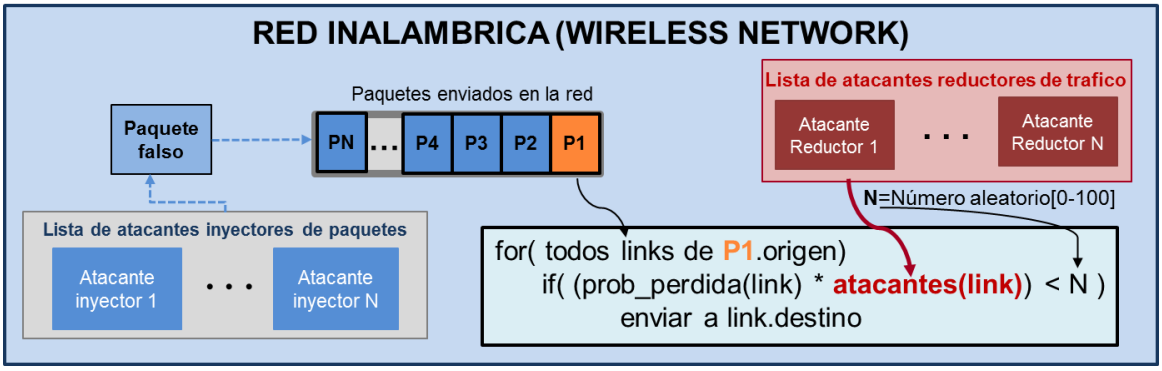


Figura 35: Simulación con atacantes inyectores y reductores del tráfico

3.6.4. Ataques que modifican el firmware/Hardware de un nodo

El atacante directo modifica el software (firmware) del nodo atacado. El modelo implementado en el simulador sustituye la programación de un nodo genuino por un programa falso. La definición del ataque incluye la nueva aplicación que será cargada en el nodo atacado y sus parámetros de red ("*packet-loss probabilities*"). En definitiva, lo que se hace es modificar el firmware de un nodo genuino por el software maligno y, a continuación, simular normalmente.

3.7. Evaluación de la simulación de redes con ataques

En este apartado se van a presentar diversos ejemplos del funcionamiento del simulador de redes de sensores inalámbricas con capacidad de emular ataques. Para evaluar el entorno desarrollado se utilizará un caso de uso que incluye distintas topologías de red (Sección 3.7.2). Para facilitar el uso del simulador de redes con ataques, se ha desarrollado una interfaz gráfica que simplifica su uso y configuración. Dicha interfaz será descrita en la siguiente sección.

3.7.1. Interfaz del simulador

Describir una red de sensores inalámbrica es una tarea de complejidad media. Sin embargo, esta tarea se complica enormemente cuando, además de describir la red, es necesario especificar los posibles ataques que dicha red puede sufrir. A la hora de especificar los ataques, el diseñador se enfrenta a dos problemas. En primer lugar, debe determinar qué ataques deben ser tenidos en cuenta y a qué nodos de la red afectan. En segundo lugar, debe determinar la configuración específica de cada ataque.

Para ayudar al diseñador en esta tarea se ha desarrollado una interfaz gráfica que permite definir todos los parámetros de la red a simular, desde la arquitectura HW y SW de cada nodo hasta la topología de red. Además, la interfaz permite definir y configurar los atacantes que modelan ataques a la red. En la Figura 36, se puede ver una captura de pantalla de la interfaz del simulador. Desde dicha interfaz es posible configurar el modelo de red de forma sencilla, así como indicar el código software y las características hardware de cada nodo. Además, se pueden añadir y configurar los atacantes que se precisen.

Los resultados de las estimaciones obtenidas para cada uno de los nodos de la red (por ejemplo, energía consumida) se muestran en la interfaz conforme avanza la simulación. Al mismo tiempo, la interfaz presenta estadísticas del uso de red, como el número de paquetes transmitidos y recibidos por cada nodo. En la parte inferior de la Figura 36 es posible apreciar una pequeña ventana con los informes finales generados por el entorno de análisis de prestaciones. Aunque esta interfaz también podría ser utilizada en el caso de simulación de redes en las que no haya riesgos de seguridad, las mayores ventajas se obtienen cuando es necesario especificar ataques y evaluar su impacto en prestaciones.

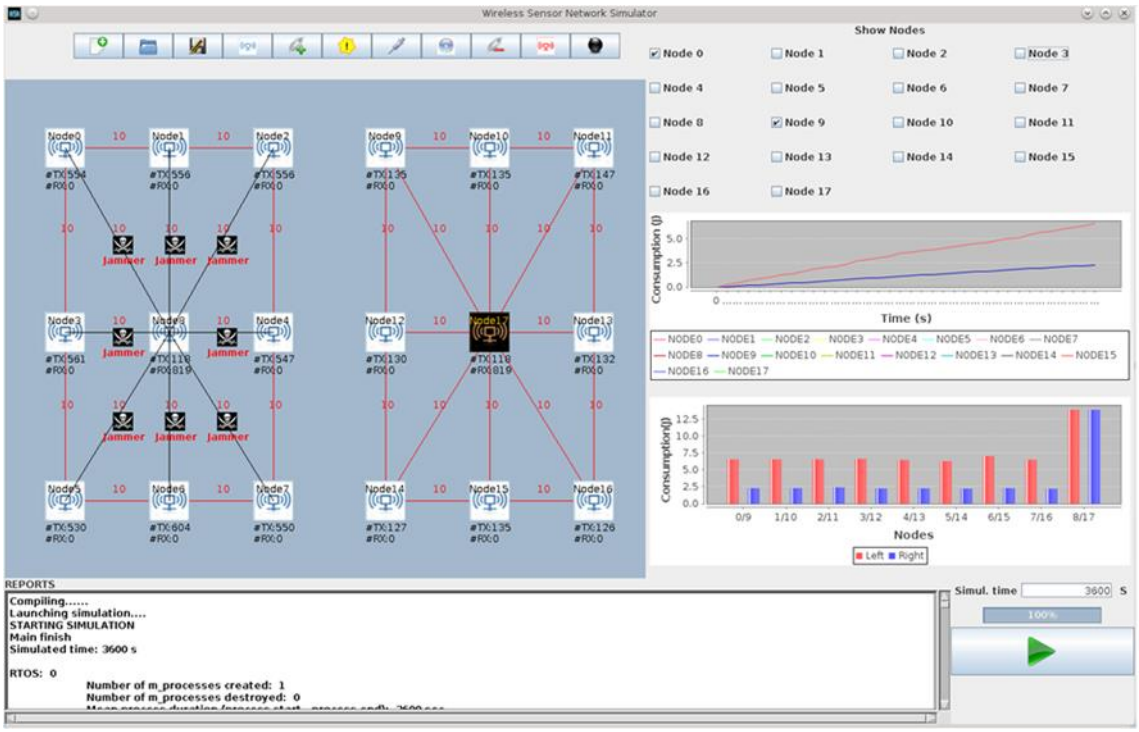


Figura 36: Captura de pantalla de la interfaz del simulador

La Figura 36 muestra la simulación de dos redes iguales (ventana superior izquierda), con el mismo número de nodos y la misma topología de la red, siendo el mismo el hardware y software de cada nodo equivalente. La diferencia entre la red de la derecha y de la izquierda, es que en la primera se ha insertado un ataque de tipo *jammer*, por lo que el consumo y prestaciones de dicha red son peores. En la parte superior-derecha de la figura, se pueden seleccionar los nodos a representar gráficamente (sección "*Show Nodes*"). Para cada nodo seleccionado se muestra la energía consumida durante todo el tiempo de simulación. En la gráfica de la parte inferior-derecha se puede observar el consumo de cada nodo en cada red (color rojo para los nodos atacados versus azul para los nodos de la red sin ataques). Esta información facilita enormemente el análisis del impacto de los ataques y reduce el tiempo de desarrollo.

3.7.2. Caso de uso de simulación de ataques

3.7.2.1. Descripción del escenario a simular

En esta sección se presenta un ejemplo de uso del simulador, en el cual se pone el acento en su capacidad para modelar y simular ataques. Además, se muestran las ventajas que ofrece el simulador para identificar tanto la topología más eficiente de la red de sensores inalámbrica como la mejor arquitectura de nodo. Como el objetivo de esta sección es la evaluación de la simulación de ataques, el análisis realizado con el simulador se centrará en cómo los ataques afectan a las propiedades de los nodos, en especial a su consumo de potencia. Los nodos de las redes de sensores inalámbricas son dispositivos alimentados con baterías, siendo uno de los principales efectos que producen los ataques el incremento del consumo de energía, lo que puede reducir drásticamente la duración de la batería y, con ello, la vida operativa del nodo y de la red.

El caso de uso evalúa una red que monitoriza una propiedad física: la temperatura ambiente. Para ello, los sensores de la red adquirirán información periódicamente, tomando una muestra cada 3 segundos. Los datos de los nodos sensores serán recopilados por un nodo Gateway, que será el encargado de analizarlos. Dependiendo de los resultados de este cómputo, el nodo Gateway tomará distintas decisiones sobre la integridad de la instalación controlada.

3.7.2.2. Exploración de distintas configuraciones de red

Para seleccionar la configuración de la red óptima, se van a tener en cuenta tres variables distintas: los tipos de nodos usados en la red, la topología de la misma y la configuración de los nodos. Los tipos de los nodos son definidos por su arquitectura hardware y software. La arquitectura hardware de los nodos (número de procesadores, tipos de baterías, etc.) tiene un gran impacto en el diseño de las redes, ya que limita la capacidad de cómputo y funcionalidad de los nodos, al tiempo que determina el coste final o presupuesto del sistema.

Las redes que se van a estudiar en esta caso de uso incluyen dos tipos distintos de nodos: el Gateway (nodo central) y los “nodos sensores” (nodos finales). La arquitectura HW/SW de cada tipo de nodo se define utilizando modelos UML [94] en las figuras adjuntas: Figura 37 (Gateway) y Figura 38 (nodos sensores). El nodo Gateway es el responsable de coordinar y controlar la red, así como de la comunicación con redes externas. El nodo sensor incluye un módulo encargado de leer la temperatura ambiente. Como es habitual en este tipo de redes, los nodos pasan a un estado *sleep* (dormido) cuando terminan sus operaciones (modo activo) con objeto de reducir su consumo e incrementar la vida de la batería.

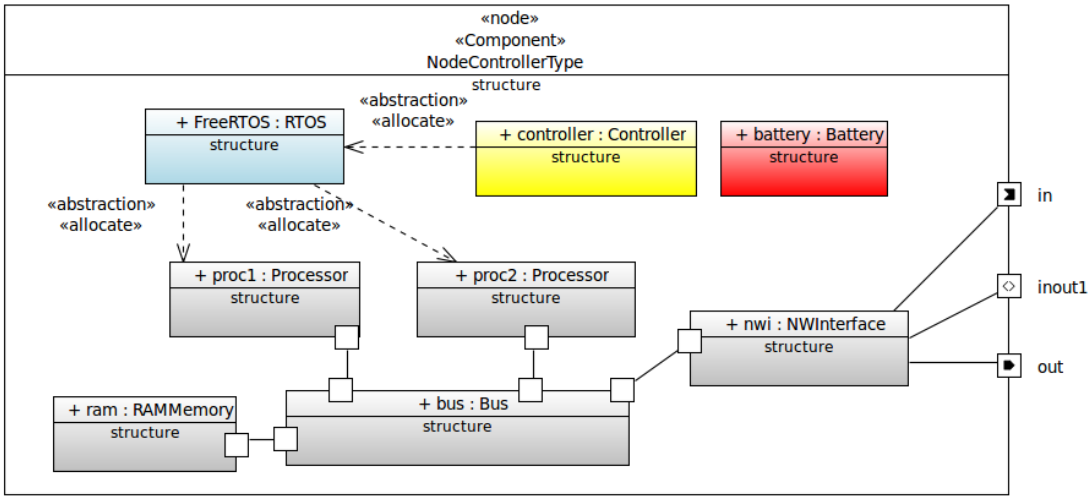


Figura 37: Arquitectura HW/SW del nodo Gateway

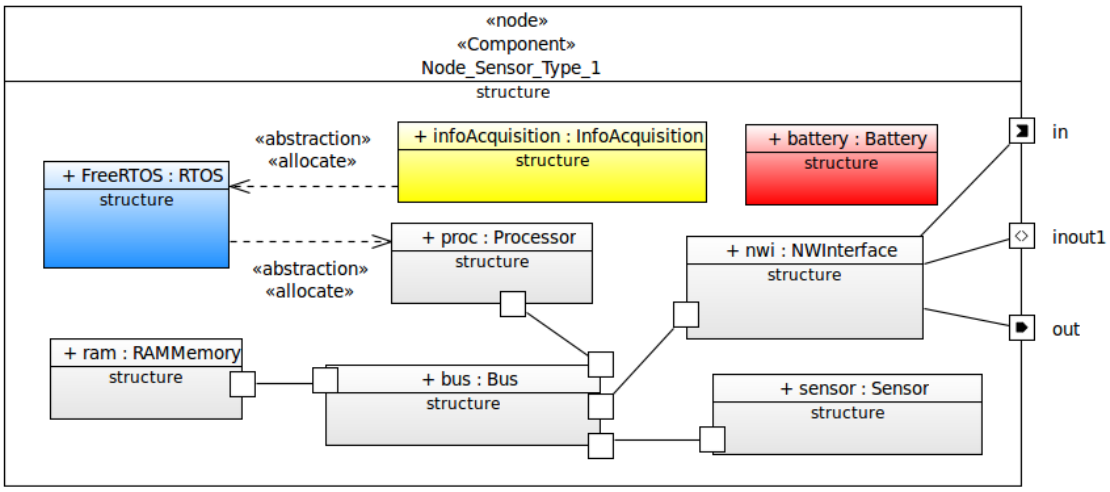


Figura 38: Arquitectura HW/SW de los nodos sensores

Cada tipo de nodo (*Gateway* y sensor) ejecuta un software embebido distinto. Sin embargo, todos integran el mismo RTOS: FreeRTOS. La funcionalidad básica de cada tipo de nodo se comenta brevemente a continuación:

- **Gateway**: Es el responsable de recibir los mensajes de los nodos sensores y transmitir la información del sistema a una red externa. Cuando los nodos sensores están activos, el Gateway espera recibir las temperaturas de todos los nodos sensores. Cuando ha recogido todos los datos de la red, genera un único mensaje y lo envía a través de un módulo GPRS a Internet. Después de esto, el nodo pasa a un estado de bajo consumo o dormido (*sleep)* durante 3 segundos.

- **Sensor**: Después de que el nodo pase a estado activo (nodo despierto), el programa lee el sensor de temperatura y envía sus datos al Gateway o a otro nodo sensor que pueda transferir la información al Gateway. Cuando finaliza su función, se desactiva (pasa a estado dormido o *sleep*) durante 3 segundos. Por lo tanto, la frecuencia de adquisición de datos es cada 3 segundos.

Además se asume que cada despliegue de la red tendrá unas características físicas distintas, que afectarán a la comunicación entre los nodos. Estas características del canal de comunicación inalámbrico se modelarán mediante distintas probabilidades de éxito en la comunicación entre nodos. Adicionalmente, se van a considerar para cada nodo y área de instalación unos valores distintos de configuración del módulo de RF (*transceiver*), lo que afectará a la potencia de transmisión, número de reintentos y encriptación.

3.7.2.2.1. Exploración de la topología de red

Los primeros experimentos evalúan los despliegues de red presentados en figuras de la subsección: Figura 39, Figura 40 y Figura 41.

La Figura 39 muestra una topología de red lineal, donde el nodo de un extremo transmite su información al nodo siguiente, el cual recoge esa información y la

encapsula, junto a sus propias lecturas, para enviárselas al nodo anterior en la cadena. Este proceso se repite hasta que toda la información llega al Gateway.

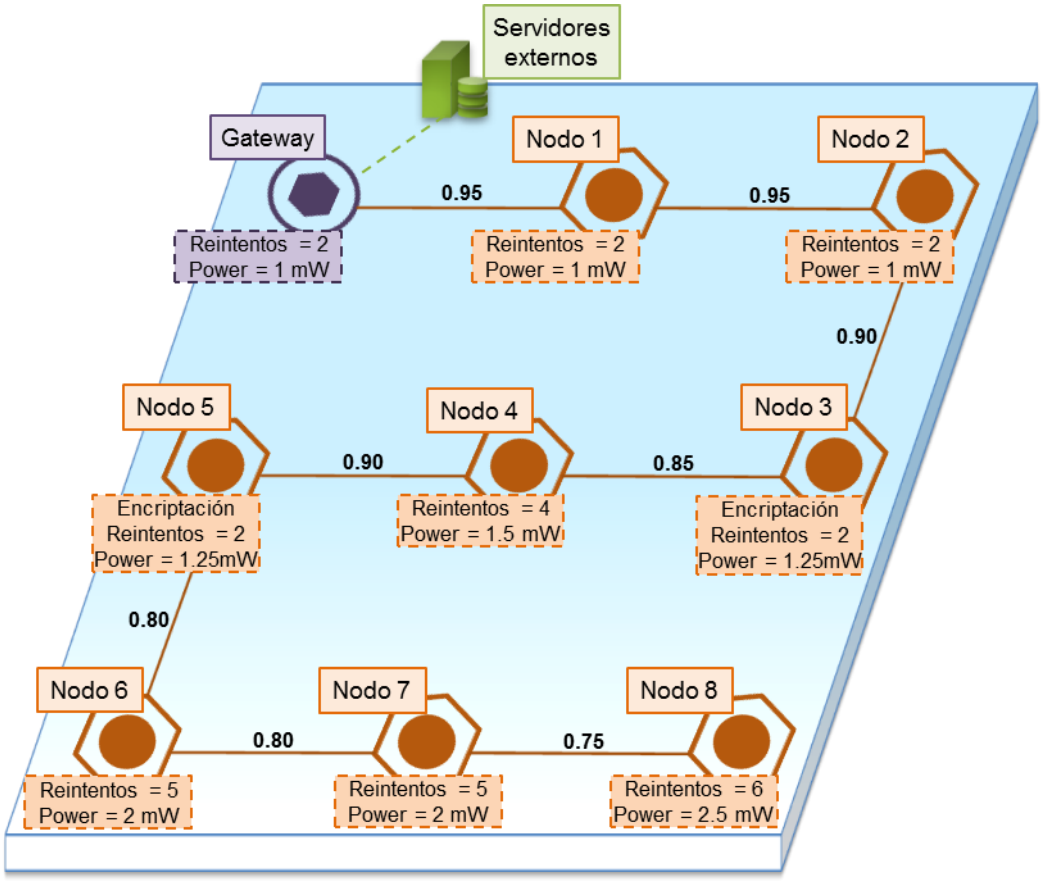


Figura 39: Topología de la red lineal

La Figura 40 muestra una topología con formato de estrella, en donde todos los nodos sensores están directamente conectados (tienen visibilidad) con el nodo Gateway. En este caso, los nodos sensores no comparten información entre ellos y envían sus datos directamente al nodo central.

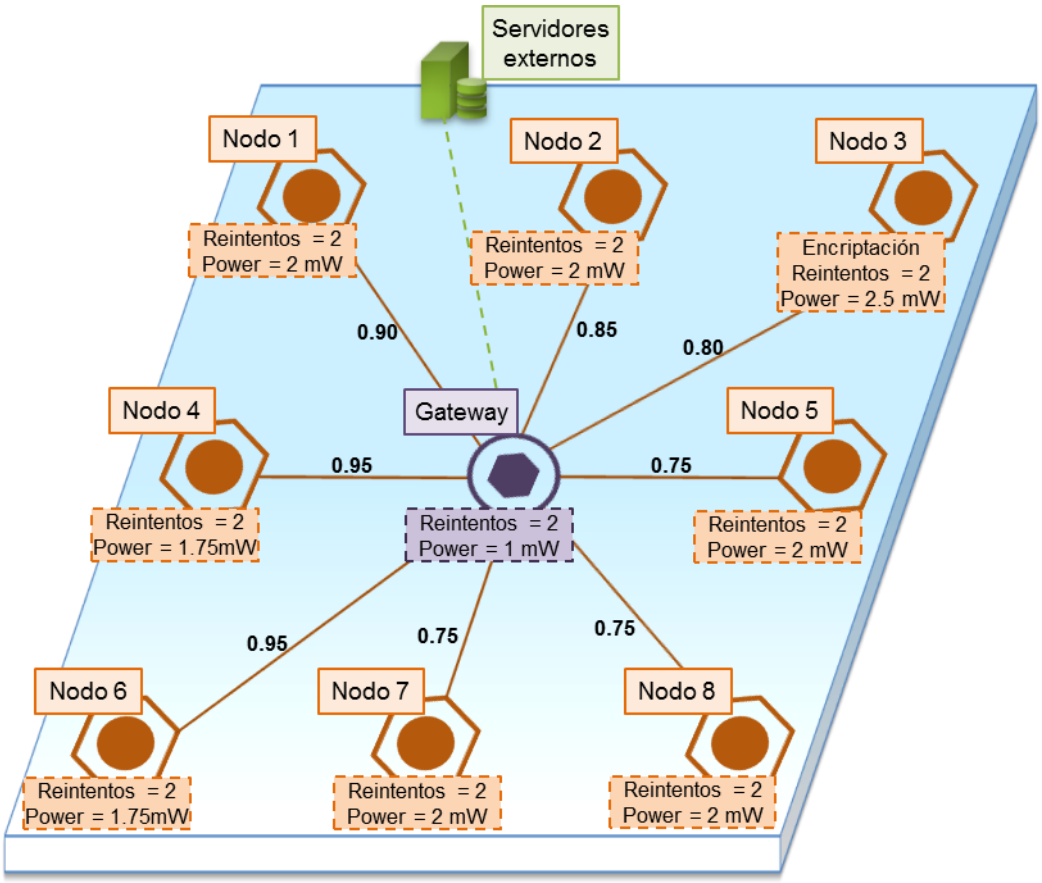


Figura 40: Topología de la red en estrella

En el último caso, presentado en la Figura 41, se presenta una topología mixta, en donde hay nodos conectados directamente al nodo Gateway (topología de estrella) junto con otros conectados en forma de árbol.

Además de la topología de red, se va a evaluar el impacto de posibles ataques que la red pueda sufrir. Por esta razón se añadirán diferentes tipos de ataques a las redes descritas anteriormente, seleccionado nodos que juegan un rol importante en el sistema como elementos a ser atacados. Por ejemplo, el nodo 3 de la Figura 41 genera información importante que puede tener un alto impacto en la seguridad de la red (monitoriza una zona crítica). Por ello, las comunicaciones de ese nodo se han encriptado, lo cual aumentará el consumo del sistema debido al proceso de encriptación.

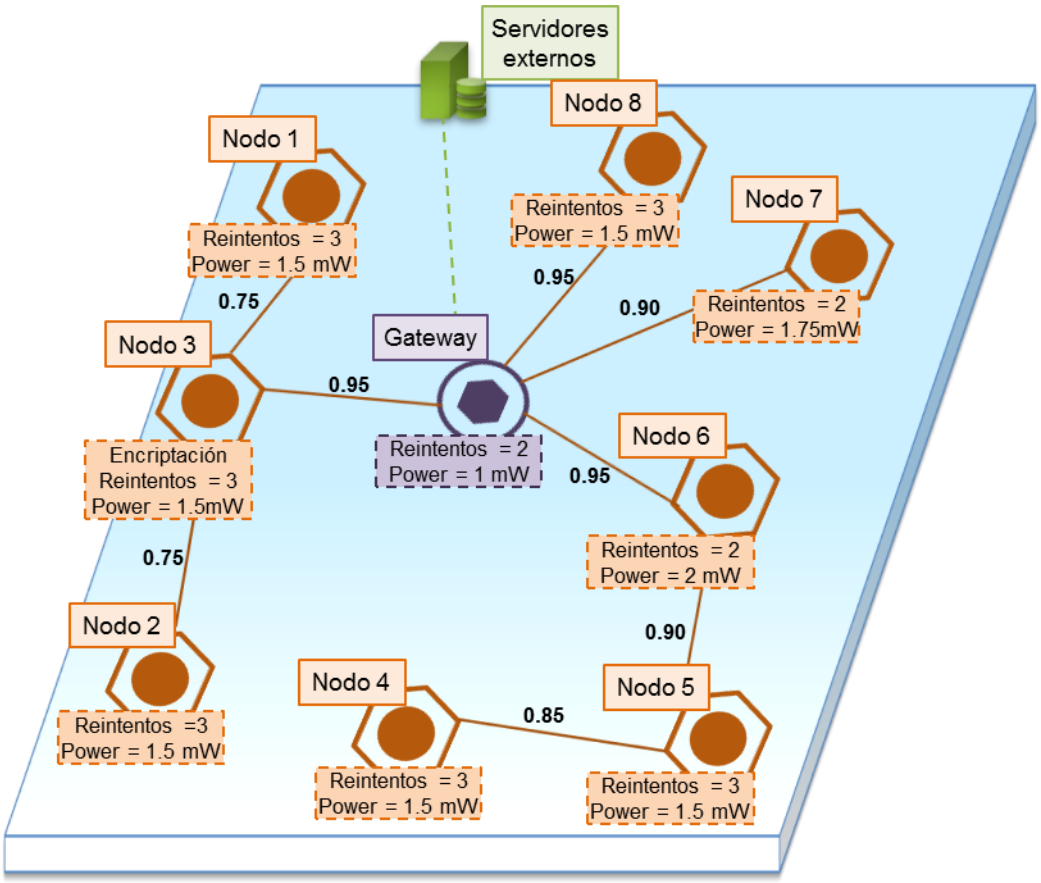


Figura 41: Topología de la red mixta e irregular

En las pruebas, el nodo 3 será atacado mediante un “ataque por colisión” (Sección 3.3.1.2), lo que provoca que el nodo no pueda enviar sus datos, afectando la integridad de la red completa.

Otro nodo seleccionado para ser atacado es el nodo 7. En este caso, se implementará un ataque por *Hello Flood* (Sección 3.3.2.3), el cual trata de agotar la energía de la batería del nodo.

Finalmente, el último nodo atacado será el nodo 6. En este caso, se ha modelado un ataque L*ooping in the network* (Sección 3.3.3.4) que afectará a un nodo importante tanto en la red lineal como en la mixta, ya que canaliza información de otros nodos (Figura 39 y Figura 41). De este modo, si este nodo se inutiliza, indirectamente se inutilizan otros nodos de la red.

Para cada topología de red (Figura 39, Figura 40 y Figura 41) se han simulado los tres ataques anteriormente descritos. Cada tipo de ataque se ha introducido en cada tipo de red con los mismos valores de los parámetros de configuración descritos en la Sección 3.4.

Se ha seleccionado el ataque por colisión como caso de ataque que introduce ruido en la red. Para ello se han definido tres atacantes que introducen tráfico en las tres topologías consideradas, todos ellos con la misma configuración: potencia del 60%, con el 100% de los paquetes afectados y alterando cualquier tipo de paquete. La única diferencia entre los 3 atacantes son los enlaces a los que afectan, ya que atacan al mismo nodo pero en topologías diferentes. Como ejemplo de ataques que introducen paquetes en la red se ha seleccionado un ataque por *Hello Flood* al nodo 7, que ha sido implementado de forma similar a la descrita en el caso anterior. Finalmente, como ejemplo de ataques que introducen ruido y paquetes a la red, se ha seleccionado el ataque *Looping in the network* al nodo 6. En este caso se definen dos atacantes distintos: uno inyector de ruido en la red (que corta la comunicación de los enlaces por los que transmite paquetes el nodo 6) y otro inyector de paquetes falsos (que introduce paquetes de datos falsos para modelar el bucle).

3.7.2.2.2. Resultados de la simulación

Se han simulado las tres redes presentadas en el apartado anterior durante 5 horas de tiempo virtual (tiempo simulado). Se han considerado 4 situaciones: sin ningún ataque o siendo afectada por uno de los 3 ataques presentados. Las estimaciones del consumo de los nodos obtenidas mediante las simulaciones para cada red y situación se presentan en la Tabla 9 (red lineal), la Tabla 10 (red en estrella) y la Tabla 11 (red mixta e irregular).

Tabla 9: Consumo (en Julios) de la red lineal

	No atacada	Collision	Hello Flood	Looping
Gateway	33.82 J	33.82 J	33.82 J	33.82 J
Nodo 1	93.65 J	93.65 J	93.65 J	93.65 J
Nodo 2	93.65 J	93.65 J	93.65 J	93.65 J
Nodo 3	107.33 J	141.53 J	107.33 J	107.33 J
Nodo 4	122.15 J	122.15 J	122.15 J	122.15 J
Nodo 5	107.33 J	107.33 J	107.33 J	107.33 J
Nodo 6	150.65 J	150.65 J	150.65 J	222.88 J
Nodo 7	150.65 J	150.65 J	255.05 J	150.65 J
Nodo 8	151.17 J	151.17 J	151.17 J	151.17 J

Tabla 10: Consumo (en Julios) de la red en estrella

	No atacada	Collision	Hello Flood	Looping
Gateway	216.86 J	216.86 J	216.86 J	198.03 J
Nodo 1	66.59 J	66.59 J	66.59 J	66.59 J
Nodo 2	68.96 J	68.96 J	68.96 J	68.96 J
Nodo 3	84.43 J	123.7 J	84.43 J	84.43 J
Nodo 4	58.55 J	58.55 J	58.55 J	58.55 J
Nodo 5	73.72 J	73.72 J	73.72 J	73.72 J
Nodo 6	58.55 J	58.55 J	58.55 J	58.55 J
Nodo 7	73.72 J	73.72 J	178.1 J	73.72 J
Nodo 8	73.72 J	73.72 J	73.72 J	73.72 J

Tabla 11: Consumo (en Julios) de la red mixta e irregular

	No atacada	Collision	Hello Flood	Looping
Gateway	141.56 J	124.24 J	141.56 J	118.19 J
Nodo 1	110.84 J	110.84 J	110.84 J	110.84 J
Nodo 2	120.48 J	120.48 J	120.48 J	120.48 J
Nodo 3	124.32 J	168.71 J	124.32 J	124.32 J
Nodo 4	105.57 J	105.57 J	105.57 J	105.57 J
Nodo 5	118.73 J	118.73 J	118.73 J	118.73 J
Nodo 6	110.98 J	110.98 J	110.98 J	153.43 J
Nodo 7	60.64 J	60.64 J	151.99 J	60.64 J
Nodo 8	56.47 J	56.47 J	56.47 J	56.47 J

Como se ha comentado anteriormente, para cada tipo de red se han simulado los tres ataques por separado. En la primera columna de cada tabla se presentan los consumos de cada nodo en el caso de que la red no sea atacada. En las siguientes columnas se pueden ver los consumos estimados de cada nodo cuando la red es atacada mediante una ataque por colisión (columna 2), *Hello Flood* (columna 3) y *Looping in the network* (última columna).

Como se puede apreciar en las primeras columnas de las tablas anteriores o en la Figura 42, el consumo del nodo Gateway en la red lineal es menor que en las otras topologías. Esto es debido a que en esa topología el Gateway recibe un número menor de paquetes, al recibir toda la información en un único bloque encapsulado por el resto de los nodos de la red. Sin embargo, el consumo de los demás nodos es mayor que en los otros casos debido al incremento del trabajo de cada nodo y al número de paquetes transmitidos. Además, en la Figura 42 se puede apreciar que la diferencia de consumo entre nodos es menor en la red mixta o irregular que en otras topologías, lo que implica una mejor distribución de tareas entre nodos.

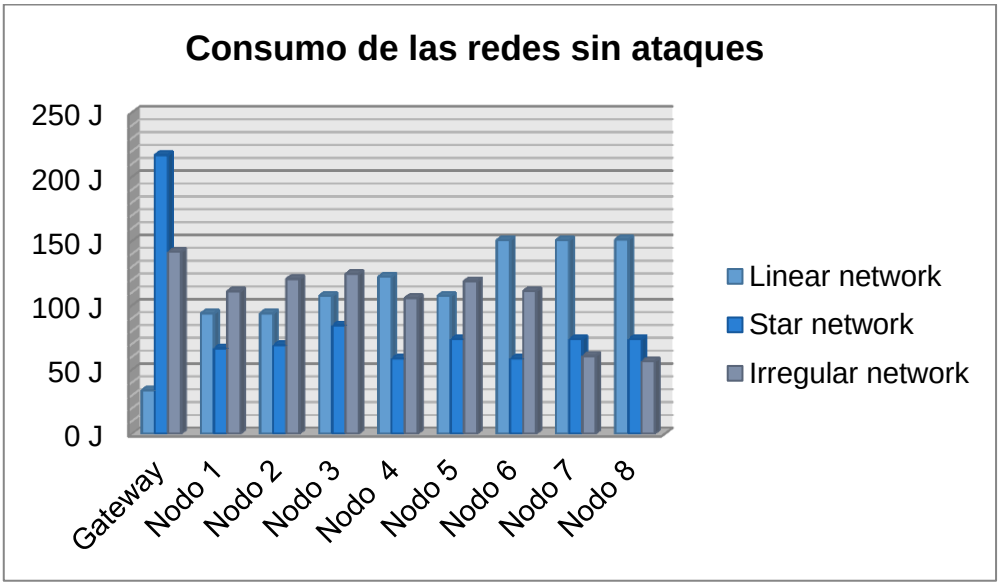


Figura 42: Consumo de los nodos en las redes sin atacar

Además de evaluar el impacto de la topología en el consumo de los nodos, la simulación permite estudiar el efecto de los ataques. Dicho efecto se puede observar en la Figura 43.

En el caso de ataque por colisión al nodo 3, se puede apreciar como dicho nodo incrementa su consumo entre un 30% y un 46%, dependiendo de la topología de red. En este caso, la red más vulnerable (en términos de consumo) es la red en estrella, en la que el nodo 3 incrementa su consumo un 46%, lo que reduce el ciclo de vida del nodo significativamente. También se puede observar que, en la red irregular, el consumo del Gateway se reduce cuando está siendo atacada. Esto es debido a que el Gateway reduce su carga de trabajo al dejar de recibir los paquetes del nodo 3 durante el ataque. La reducción de paquetes no solo es debida al ataque sino que también es consecuencia de la configuración del nodo, que solo permite 2 reintentos de envió de paquetes fallidos. En las otras topologías, al estar configurado el nodo con un número de reintentos mayor, se consigue que el ataque sea menos efectivo ya que la gran mayoría de los paquetes son recibidos por el Gateway, aunque con un incremento del consumo de energía.

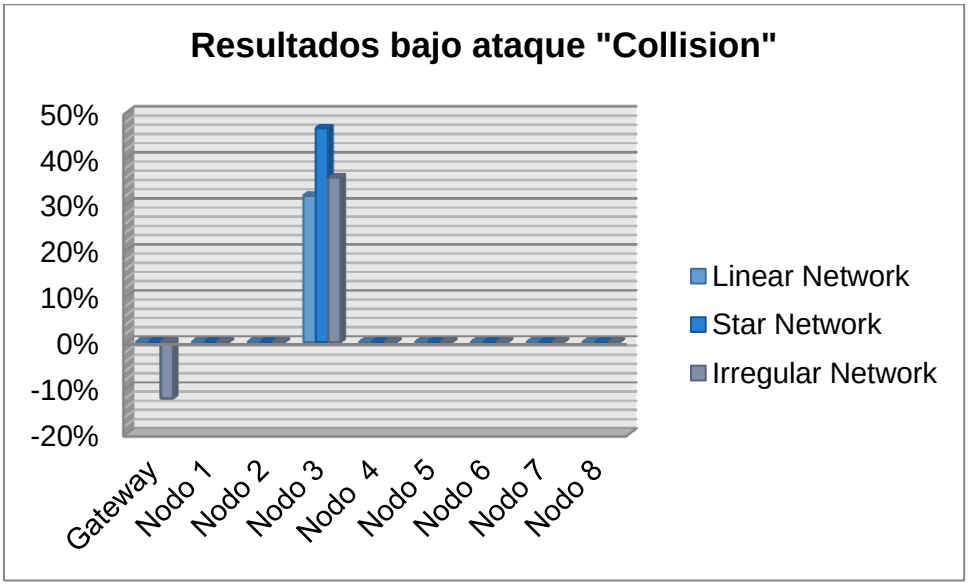


Figura 43: Consumo de los nodos en las redes bajo ataques por colisión

Como se puede observar en la Figura 44, el ataque *Hello Flood* aumenta significativamente el consumo de energía de los nodos atacados (desde el 69% hasta el 150%, en el peor caso). Por lo tanto, este ataque causa mucho daño a las redes ya que el drástico incremento de consumo implica que el ciclo de vida de los nodos atacados se reduce notablemente. Esto obliga a sustituir con mayor frecuencia la batería de los nodos, con el consiguiente incremento de costes.

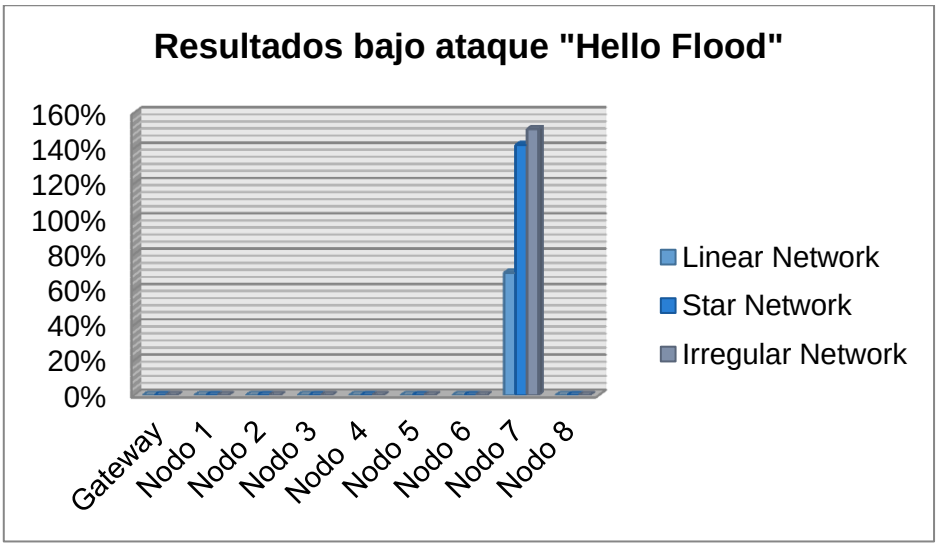


Figura 44: Consumo de los nodos bajo ataques por Hello Flood

Finalmente, la Figura 45 presenta los resultados de la simulación de las redes bajo el ataque *Looping in he network*. Como se puede observar en las tablas, este ataque afecta a las tres redes más o menos de manera similar. Sin embargo, en el caso de la red irregular o en estrella, el nodo Gateway recibe menos paquetes de los que recibiría en el caso de no existir ataques, lo que le podría llevar a tomar decisiones incorrectas por falta de información. En el caso de la red lineal, el nodo Gateway recibe el mismo número de paquetes que si no estuviese siendo atacada. Sin embargo, estos paquetes estarán incompletos debido a la falta de información del nodo 6 y sus antecesores.

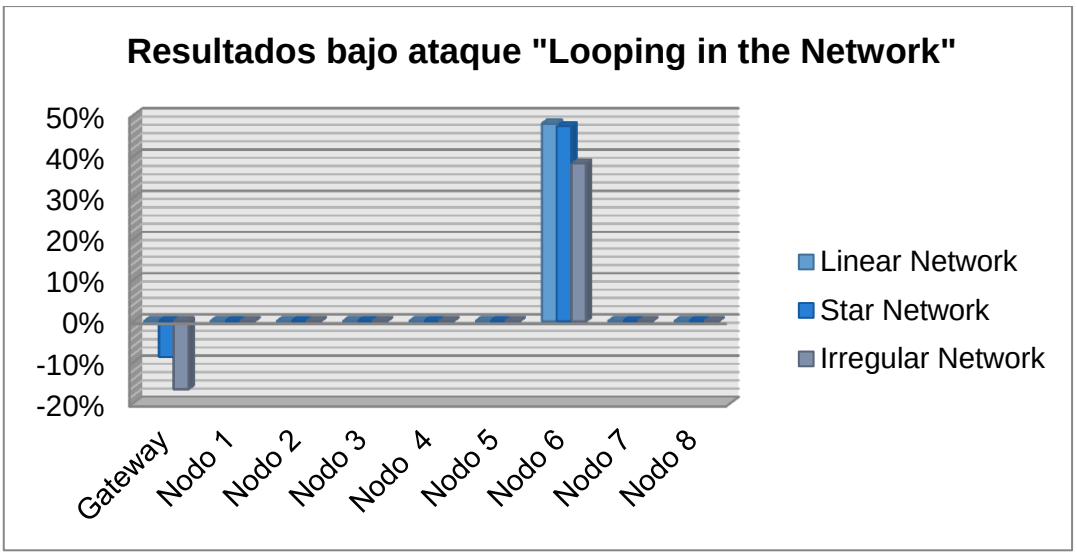


Figura 45: Resultados de los consumos de los nodos bajo ataques por Looping

3.7.2.2.3. Otros Resultados: Exploración de la configuración de los nodos

Se han llevado a cabo una serie de simulaciones en las cuales se ha modificado la configuración de los nodos en una red con topología irregular (Figura 41). La modificación afecta a la potencia de transmisión del módulo (*transceiver*) de RF, lo que implica un cambio en las probabilidades de transmisión correcta. Específicamente se han modificado la potencia y las probabilidades de los enlaces de los nodos 3 y 6. Las modificaciones se pueden apreciar en la Tabla 12 y Tabla 13. Además, en ambas tablas se puede observar el consumo estimado en cada caso. Con el objetivo de asegurar la correcta transmisión de los datos, cada nodo se ha configurado con un relativamente alto número de reintentos: cuatro reintentos en el caso del nodo 3 y tres reintentos en el caso del nodo 6.

Tabla 12: Resultados simulados de los consumos del nodo 3 en Julios

Potencia de transmisión (mW)	**1**	**1.5**	**2**	**2.5**	**3**
Configuración del nodo (probabilidad en %)	50	55	90	92	93
Consumo (J)	217.22	183.84	161.59	186.62	212.21

Tabla 13: Resultados simulados de los consumos del nodo 6 en Julios

Potencia de transmisión (mW)	**1**	**1.5**	**2**	**2.5**	**3**
Configuración del nodo (probabilidad en %)	60	64	70	78	95
Consumo (J)	146.08	136.51	159.76	170.02	166.60

Como se puede observar en la Figura 46, los resultados muestran que el consumo de cada nodo varía con la potencia de transmisión. Inicialmente se podría esperar que una potencia de transmisión mayor implicara un aumento del consumo de energía. Sin embargo, esto no es del todo cierto debido a que el aumento de potencia lleva implícita

una variación de la probabilidad de recibir los paquetes y, por lo tanto, una reducción en el número de reintentos utilizados. Por lo tanto, para obtener la configuración óptima es necesario balancear la potencia de transmisión y la probabilidad de recibir el paquete. En el caso del nodo 3, cuando la potencia de transmisión es de 1mW, la probabilidad de recibir el paquete es baja, por lo que se necesitan más reintentos para recibir el paquete con el consiguiente incremento de consumo. Cuando la potencia de transmisión es de 3mW el consumo del nodo aumenta aunque la probabilidad de recibir el paquete es mayor, lo que reduce el número de reintentos y, por ello, el cómputo y consumo. Por lo tanto, de acuerdo con la Figura 46, la potencia de transmisión óptima para dicho sistema es de 2mW. Con dicha potencia se observa un valle del consumo, ya que el incremento de consumo producido por el incremento de potencia de transmisión se compensa con la reducción del número de reintentos necesarios para transmitir el mensaje.

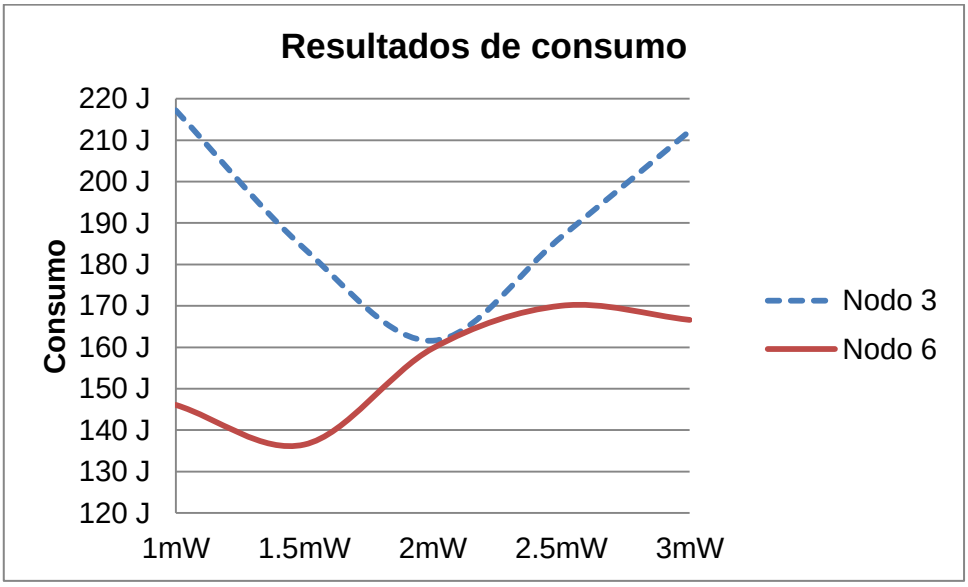


Figura 46: Consumo de energía de los nodos 3 y 6

A continuación se analiza el efecto de un ataque por Jamming (Sección 3.3.1.1) que afecte a la comunicación entre los nodos 3 y 6. Para cada caso (potencia de transmisión de 1mW, 1,5mW, etc.) el *jammer* introduce una probabilidad de error distinta. Por ejemplo, en el caso de 1mW, el ruido introducido por el *jammer* es suficiente para afectar al 100% de los paquetes transmitidos. Sin embargo, en el caso de 3mW, la potencia del *jammer* no es suficiente para afectar a todos los paquetes y únicamente se ven afectados un 20% de ellos.

Tabla 14: Resultados simulados de los consumos de los nodos 3 y 6

Potencia de transmisión(mW)	**1**	**1.5**	**2**	**2.5**	**3**
Efectividad del ataque Jamming (Probabilidad en %)	100	90	70	40	20
Consumo del nodo 3 (J)	272.85	342.39	384.12	293.71	250.603
Consumo del nodo 6 (J)	187.12	235.01	269.21	238.42	211.75

En la Tabla 14 y la Figura 47 se pueden ver los consumos de los nodos cuando la red es afectada por un *jammer*. En este caso, la configuración óptima para el nodo 3 fija su potencia de transmisión a 3mW. Comparando este resultado con el mostrado en la Figura 46 se puede observar que la potencia seleccionada en dicha figura (2mW en el caso de no existir ataques) es la que peores resultados obtiene en caso de ataque. Esto es consecuencia del alto porcentaje de paquetes perdidos debido a que, con solo 2mW de potencia, la transmisión está muy afectada por el *jammer*.

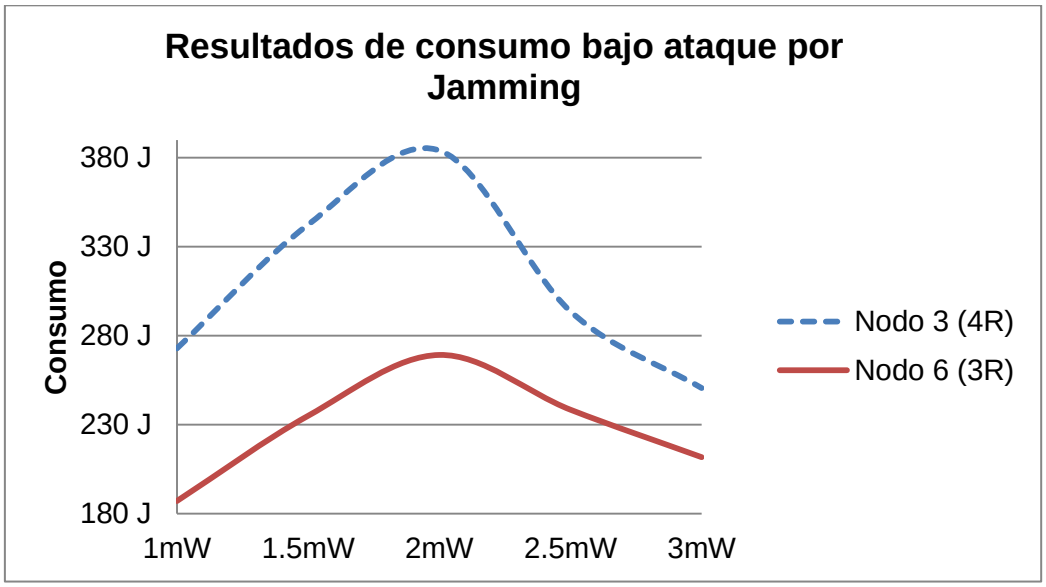


Figura 47: Consumo de energía de los nodos 3 y 6 bajo ataque Jamming

Teniendo en cuenta los resultados mostrados en las figuras anteriores (Figura 46 y la Figura 47), el diseñador puede tomar decisiones óptimas sobre la configuración de los nodos para una topología concreta.

3.8. Diseño de firmware contra ataques

Como se ha comentado anteriormente, las características de las redes de sensores inalámbricas (como el ser sistemas autónomos que operan en entornos hostiles) las hacen especialmente vulnerables a ataques. Por esta razón, mejorar la seguridad es uno de los requisitos cada vez más común en las especificaciones de este tipo de sistemas. El nivel de seguridad requerido puede variar, dependiendo de la importancia de los datos que se van a obtener o intercambiar. Sin embargo, en todos los casos es crucial identificar las debilidades en la seguridad de la red durante la fase de diseño.

Una forma de mejorar la seguridad es conocer los efectos que los ataques más típicos pueden provocar en un nodo (o a toda la red), lo que permite prevenir futuras vulnerabilidades del sistema. Para lograr este objetivo, el simulador presentado en este capítulo puede jugar un papel esencial. Dicha herramienta tiene capacidad para proveer estimaciones de tiempos de ejecución y consumo, mientras proporciona soporte para el RTOS, lo que permite analizar la ejecución del SW real sobre una plataforma HW al tiempo que permite simular los ataques más típicos. El simulador permite a los diseñadores conocer el comportamiento del sistema, facilitando la mejora del rendimiento de los nodos en cada tipo de despliegue así como identificar los ataques más peligrosos y diseñar software embebido (firmware) específico que pueda detectar y evitar dichos ataques. El objetivo de esta sección es presentar una metodología que permita desarrollar software embebido (firmware) capaz de soportar ataques.

3.8.1. Metodología para diseñar firmware que soporte ataques

El simulador desarrollado ha permitido definir una metodología que facilita el diseño de firmware que evita ataques mediante el uso de contramedidas implementadas en software, así como comprobar su correcto funcionamiento. Hay que tener en cuenta que el uso de esta metodología no permite la generación automática de firmware que evita ataques, sino que provee al desarrollador con estimaciones que le guían a la hora de

definir medidas contra ataques. La metodología propuesta se presenta en la Figura 48 y consta básicamente de tres etapas:

1. Evaluación de los ataques.
2. Diseño de un procedimiento para detectar el ataque.
3. Diseño de contramedidas.

Como se observa en la Figura 48, en primer lugar se inyectan diferentes tipos de ataques para evaluar el comportamiento y el rendimiento de los nodos de la red. Hay que tener en cuenta que el simulador también evalúa el impacto de la topología de red y las características del canal de comunicación inalámbrico.

Las estimaciones que el simulador proporciona permiten la identificación de los ataques más dañinos y peligrosos, así como la detección de los nodos y las configuraciones de red más vulnerables. Con esta información, los desarrolladores seleccionan las contramedidas que deben ser añadidas a la red y/o los nodos cuyo firmware debe ser modificado para reducir su sensibilidad a ataques. Una vez que la contramedida es implementada, la metodología propone verificarla con nuevas simulaciones para evaluar la mejora de la seguridad. Como se observa en la Figura 48, en la fase de evaluación los desarrolladores pueden explorar y comparar los efectos de los ataques con diferentes configuraciones o contramedidas. Esto permite mejorar el rendimiento de la red mediante la modificación del software de la aplicación o el hardware de los nodos. Además, la metodología posibilita la comparación entre distintas contramedidas, por lo que solo las más eficientes serán implementadas en el sistema final, haciéndolo más seguro y robusto. Estas contramedidas pueden implementar técnicas descritas en el estado del arte o pueden ser diseñadas de forma específica con el soporte del simulador.

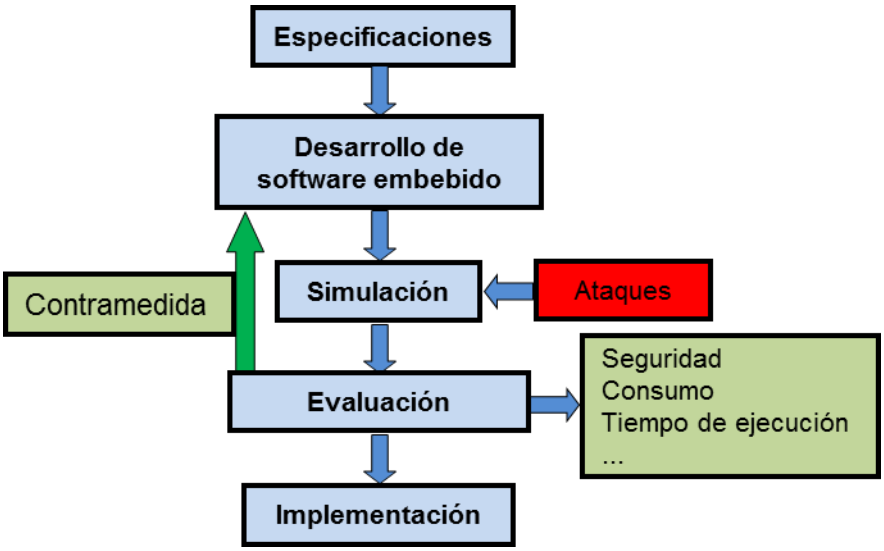


Figura 48: Diseño de firmware seguro.

3.8.1.1. Evaluación de los ataques

En esta etapa se evalúa el impacto de los ataques sobre la red o nodos específicos con objeto de detectar las agresiones más peligrosas. Los efectos de los ataques dependen de la topología de red, del software de cada nodo, de sus componentes HW e incluso de su configuración. Un ataque puede ser muy perjudicial para un tipo de nodo en una red con una topología específica pero inofensivo en otra situación. Es por ello que la simulación ayuda a identificar los ataques más problemáticos y las situaciones en las cuales su impacto es mayor. La evaluación proporciona resultados tanto funcionales (variación del comportamiento del nodo y de la red) como no-funcionales (modificación del consumo, tiempo de respuesta o ejecución). Como los ataques típicamente buscan aislar nodos de red o incrementar el consumo de energía, serán estos los principales aspectos evaluados en esta fase.

3.8.1.2. Detección de ataques en los nodos de la red

Una vez que se han identificado los ataques más peligrosos, es necesario conseguir que el firmware del nodo detecte cuando está siendo atacado. En esta tesis se propone

una técnica de detección que hace uso de las estimaciones que provee el simulador. Dichas estimaciones serán usadas para obtener perfiles de rendimiento de cada nodo de la red. Dichos perfiles serán estudiados para determinar los diferentes comportamientos que pueden producir los ataques. La idea es proveer al nodo de información de cómo debería funcionar en circunstancias normales (simulando la red sin ataques) y cómo se comportaría cuando está siendo atacado, permitiendo de esta forma detectar el ataque. Por ejemplo, si se simula la red sin ataques se puede extraer una tasa media de paquetes recibidos o enviados por cada nodo. Si se simula la misma red siendo agredida por un ataque por Jamming o por replicación, dicha tasa de paquetes recibidos variará en los nodos atacados (aumentará o disminuirá). Como la simulación de la red sin ataques puede tener en cuenta el efecto del canal y del despliegue, será posible distinguir entre ataques y modificaciones dinámicas del entorno de la red. Otro enfoque para detectar un ataque se basa en el análisis del consumo de energía del sistema [95]. Gracias a las estimaciones de consumo obtenidas con el simulador, el desarrollador podrá proveer al firmware de la capacidad de detectar cuando está siendo atacado o funcionando de manera errónea. En el caso de un ataque por Jamming, el tráfico en el nodo atacado será diferente al tráfico en condiciones normales, lo que afecta al cómputo y consumo del nodo. Debido a estas variaciones, es posible definir un rango de valores admisible en operación normal. Cuando las medidas de consumo en campo se salgan de ese rango, se podrá asumir que el nodo está siendo atacado.

3.8.1.3. Diseño de software para evitar el efecto de los ataques

Una vez que el ataque ha sido detectado, se debe diseñar una contramedida para que el mismo afecte lo menos posible al comportamiento del nodo y de la red. Los diseñadores del sistema desarrollarán dicha contramedida y la verificarán con el simulador antes de desplegar la red. Entre las técnicas normalmente utilizadas para defender al sistema se pueden destacar la desactivación (apagado) de los nodos atacados, la modificación del canal de comunicación inalámbrico, el cambio de clave de encriptación de los mensajes o incluso la exclusión del atacante de la red mediante un filtro. Las contramedidas no están limitadas a estos métodos, sino que pueden ser tan

diferentes y sofisticadas como los desarrolladores deseen. La ventaja principal de la metodología propuesta es la posibilidad de verificar la contramedida y mejorarla antes de desplegar la red, tal y como se muestra en la Figura 49. Además, la metodología nos permite comparar distintas contramedidas e implementar en los nodos desplegados solo las más eficientes.

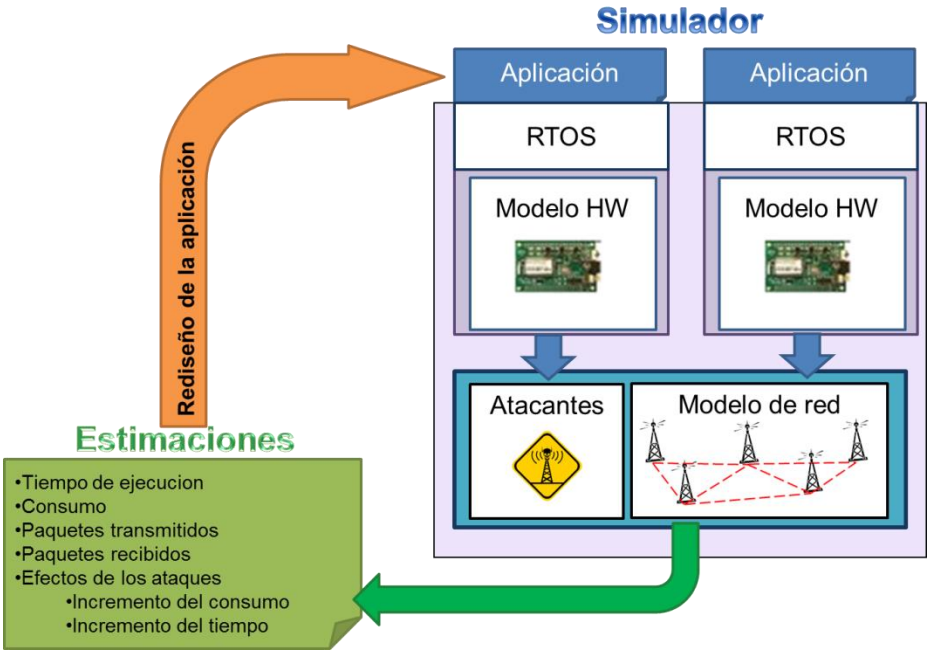


Figura 49: Diseño de firmware que evita ataques

3.8.2. Caso de uso: Análisis del ataque a una red de sensores inalámbrica

Con el objetivo de evaluar la metodología propuesta, se ha implementado y atacado una red física, con un despliegue real. El esquema de la red y del ataque se muestra en la Figura 50.

La red se compone de tres nodos: Gateway, nodo 1 y nodo 2. El Gateway es el encargado de comunicar la red con Internet, utilizando para ello un módem adicional (módulo GPRS en el ejemplo propuesto). Los otros 2 nodos realizan medidas del

ambiente con los sensores integrados en su hardware. Los tres nodos de la red tienen una arquitectura hardware similar, siendo la principal diferencia el módulo GPRS del Gateway. El hardware de los nodos utiliza un dispositivo de la familia STM32F4 con un procesador ARM Cortex-M4, memoria y varios periféricos además de un módulo de radio tipo Zigbee (802.15.4) y sensores de temperatura, presencia (PIR) y humedad.

La principal diferencia entre los nodos de la red es el software que se ejecuta en cada uno (firmware).

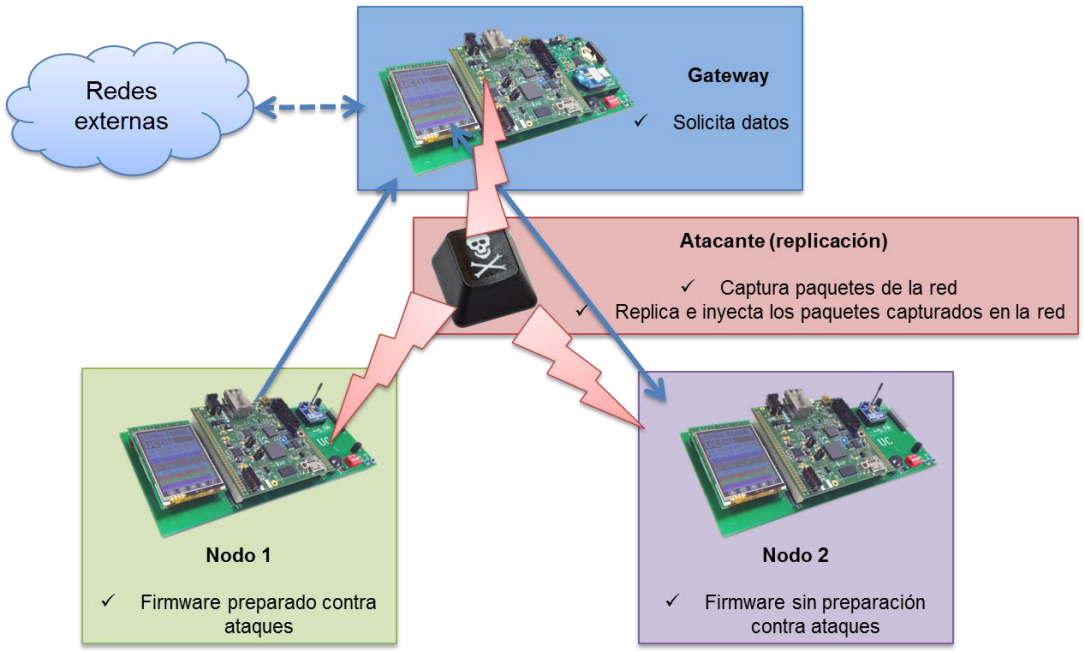


Figura 50: Red inalámbrica atacada

La aplicación del Gateway es la encargada de solicitar información cada 3 segundos a los otros dos nodos de la red. Los nodos 1 y 2 están coordinados y esperan a recibir una petición del Gateway. Cuando reciben la solicitud del Gateway, dichos nodos leen sus sensores, encriptan la información y se la envían al Gateway. Después de esto, pasan a estado inactivo (estado dormido o *sleep*) durante 3 segundos. El firmware del nodo 1 tiene capacidad de detectar y evitar ataques. Dicho firmware fue diseñado utilizando la metodología propuesta en esta sección con el objetivo de reducir los efectos de los ataques por replicación. La funcionalidad del nodo 2 es similar, aunque no incluye la protección contra ataques.

Para desarrollar el firmware del nodo 1, se ha introducido una batería de ataques en el simulador. Gracias a dicho análisis se ha podido comprobar que el ataque por replicación (Sección 3.3.3.3) podía causar graves consecuencia en el consumo de los nodos de la red, ya que sobrecargaba a los nodos y agotaba su batería rápidamente.

Una vez que la vulnerabilidad fue detectada, se inició la segunda parte de la metodología: la red fue simulada con y sin ataques, obteniéndose múltiples informes con estimaciones del comportamiento del sistema. Con esos informes se pudo estimar la tasa de paquetes recibidos por el nodo 1 tanto en el sistema atacado como cuando no se inyectaban ataques. Se observó que durante el ataque el número de paquetes recibidos se incrementaba por encima de lo normal. Esta observación permitió definir una tasa de paquetes máximos recibidos, de forma que si se sobrepasaba dicho umbral, el nodo entraba en modo de ataque detectado. En el caso de uso estudiado, la tasa de funcionamiento normal era de 1 paquete cada 3 segundos (0.33 paquetes/segundo). Si el atacante introdujese paquetes con una tasa menor a ésta, el ataque no podría ser detectado por el firmware del nodo. Sin embargo, si la tasa es mayor, el nodo detecta el ataque y activa la contramedida.

Una vez que se ha desarrollado una técnica para detectar el ataque, es necesario diseñar e implementar una contramedida efectiva. En el caso de uso se evaluaron y compararon mediante simulación distintos métodos para evitar el efecto del ataque. Se pudo observar que la técnica más efectiva era modificar el estado del nodo, pasando este a un estado de inactividad (nodo dormido) cuando el ataque es detectado. Además se definió un tiempo de inactividad específico (30 segundos) para reducir el impacto del ataque. Después de este tiempo de inactividad, el nodo se activa y comprueba si el ataque continúa, en cuyo caso se desactiva de nuevo. Por lo tanto, el objetivo principal de la contramedida propuesta es evitar que la batería se agote como consecuencia del ataque. Un esquema del comportamiento de la técnica implementada se puede observar en la Figura 51. En ella se aprecia que cuando el nodo 1 detecta el ataque, el firmware “duerme” al procesador durante un cierto tiempo. Sin embargo, en el caso del nodo 2, que no dispone de firmware protegido contra ataques, el firmware procesa todos los paquetes falsos inyectados por el atacante, lo cual provoca un incremento del consumo y una reducción de la vida de la batería.

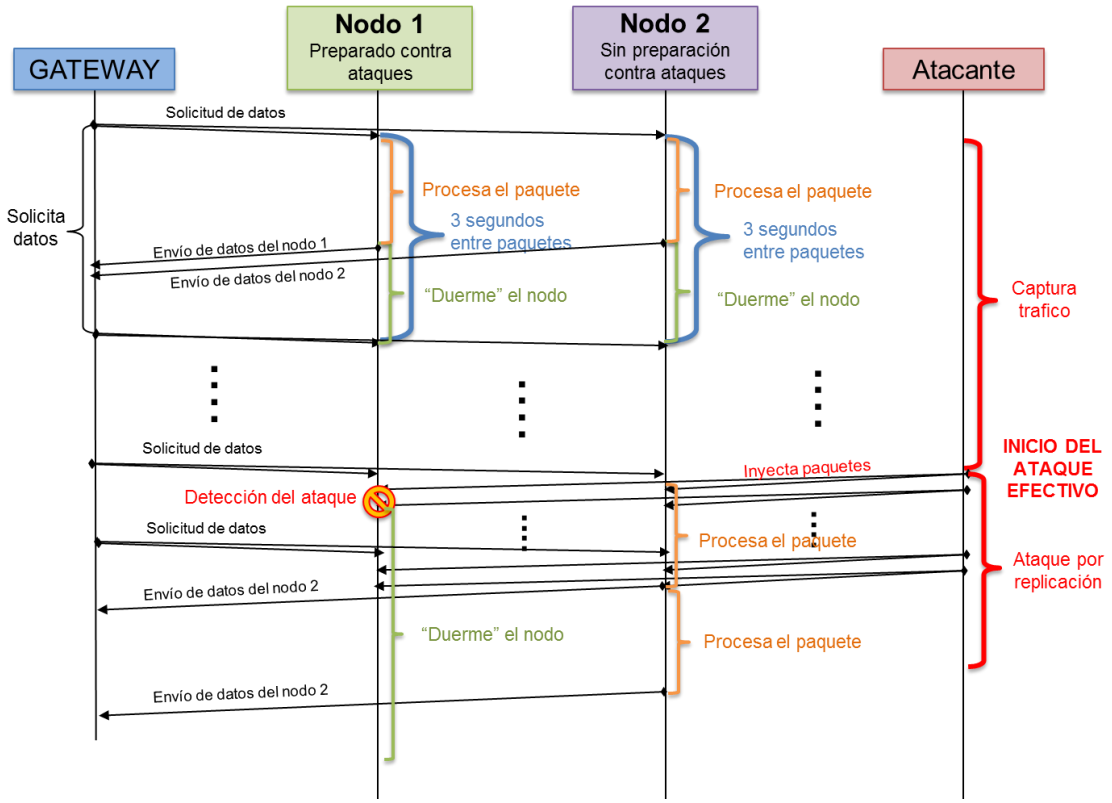


Figura 51: Ejecución de los firmwares bajo el ataque de replicación

Para validar la metodología, la red propuesta ha sido implementada físicamente (Figura 52) y se han medido los consumos reales de los nodos. El atacante se ha implementado mediante un componente adicional, el USB Raven Atmel [96], el cual se puede observar en la Figura 53. Este dispositivo integra un microcontrolador con USB y un módulo (*transceiver*) de radio. El microcontrolador recibe comandos a través del interfaz USB al tiempo que maneja el módulo de radio y los protocolos de red inalámbrica. En el caso de uso, el agresor implementa un ataque por replicación, el cual consiste en enviar repetidamente paquetes a los nodos destino. Estos paquetes son previamente capturados del tráfico real de la red por el atacante, por lo que la víctima no identifica los paquetes como falsos. Se ha configurado la agresión de forma que el atacante envíe un paquete cada 2 segundos, por lo que el ataque podrá ser detectado por el firmware diseñado para evitar el ataque.

Figura 52: Foto de la red real implementada

Figura 53: Atacante USB

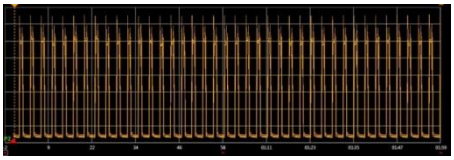
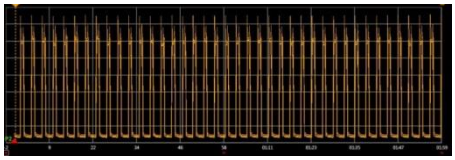
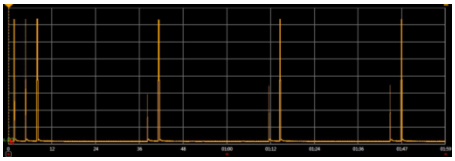
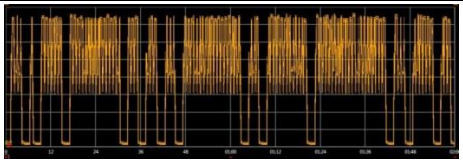
Una vez desplegada la red, se procedió a realizar mediciones reales del consumo de los nodos, para evaluar la efectividad del firmware diseñado. Estas medidas se han realizado con el Analizador de Potencia N6705b DC de Agilent [97], el cual se puede ver en la Figura 54. Este dispositivo es una fuente de alimentación diseñada específicamente para analizar el consumo del sistema y/o el comportamiento de sus baterías.

Figura 54: N6705b DC *Power Analyzer*

La Tabla 15 muestra los consumos de los nodos de la red durante un periodo de tiempo de 2 minutos y bajo distintas condiciones de operación. Se puede observar que el consumo del firmware no seguro (nodo 2) se incrementa un 221% durante el ataque. Esto supondría reducir a más de la mitad su ciclo de vida. En cambio, en el nodo cuyo firmware implementa la detección y la contramedida (nodo 1), la variación de consumo es mínima, al pasar el nodo a un estado de bajo consumo durante el ataque. Además, el consumo de ambos nodos en operación normal (sin ataques) es similar lo que demuestra que la contramedida desarrollada no incrementa el consumo durante dicho modo de operación.

En las gráficas presentadas en la Tabla 15, se puede apreciar como el nodo pasa a estado activo (picos de consumo), lee las medidas de los sensores y envía sus datos al Gateway. Cuando la red es atacada, el nodo 1 detecta el ataque y se ve como rápidamente pasa a un modo de bajo consumo durante 30 segundos (dejan de aparecer picos de consumo). Pasados estos 30 segundos, el nodo vuelve a despertar y, al ver que el ataque no ha terminado, repite la contramedida. En el caso del nodo 2, se puede apreciar como procesa todos los paquetes del agresor durante el ataque y apenas pasa a estado inactivo, lo que hace que su consumo aumente.

Tabla 15: Resultados de consumo

Estado de la red	Nodo 1 Firmware contra ataques	Nodo 1 Firmware inseguro
No atacada	 2.02 mWh	 2.02 mWh
Atacada	 1.93 mWh (-5%)	 4.48 mWh (+221%)

4. Métrica de seguridad

En las redes de sensores inalámbricas, los requisitos de seguridad no solo buscan minimizar el impacto de los ataques sino también alcanzar un nivel mínimo de confidencialidad de los datos que se manejan [98] [99] [100]. Como las redes de sensores inalámbricas utilizan un medio de comunicación compartido en un entorno no controlado, las transmisiones de datos están muy desprotegidas frente a ataques de tipo “robo de información”, como el ataque por *Sniffing* (Sección 3.3.5.1) o la alteración de paquetes. Por lo tanto, es esencial disponer de técnicas, como la encriptación de datos, que protejan al sistema frente a dichos ataques. A la hora de seleccionar un método de encriptación no solo hay que tener en cuenta el grado de confidencialidad y autenticación de los datos, sino que también es preciso valorar el consumo de energía que el cómputo del algoritmo de encriptación conlleva, ya que dicho consumo puede reducir la vida útil de la red.

El simulador presentado en el capítulo anterior permite analizar el comportamiento funcional y no-funcional de una red de sensores inalámbrica, tanto en situaciones en las cuales no hay ataques como cuando la red es atacada. Sin embargo, dicha técnica no permite evaluar la seguridad de un sistema que utiliza tramas de datos encriptados. El objetivo de este capítulo es definir una métrica que permita evaluar la seguridad de dichos sistemas y que pueda ser integrada en el simulador anteriormente descrito.

Tradicionalmente la evaluación de la seguridad de los métodos criptográficos se realiza mediante conjuntos de test que determinan cuando un algoritmo criptográfico puede ser considerado seguro. Hay múltiples trabajos comparando la seguridad de métodos criptográficos, como [101] o [102]. Sin embargo, en ellos no se define una estrategia de evaluación común que pueda ser aplicada a diferentes algoritmos criptográficos. Un ejemplo muy significativo de cómo se evalúan estos algoritmos fue la estrategia utilizada por el NIST (*National Institute of Standards and Technology*) para seleccionar el estándar de encriptación avanzado (AES). En dicha estrategia se definieron tres categorías de criterios de evaluación [102]: seguridad, coste y características de la implementación del algoritmo. Algunos de los criterios utilizados por el NIST no pueden ser evaluados fácilmente (por ejemplo la solidez de la base matemática del algoritmo). Sin embargo, hay otros criterios que pueden ser estimados mediante pruebas estándar, como por ejemplo la aleatoriedad de la salida del algoritmo o *randomness*.

La métrica que se va a integrar en el simulador está basada en la medida de la aleatoriedad del mensaje encriptado. La aleatoriedad es una propiedad común de los algoritmos criptográficos seguros de forma que la salida encriptada por dichas técnicas parece una señal completamente aleatoria. Para medir la aleatoriedad de una señal, el conjunto de test más utilizado es el "NIST *Statistical Test Suite*" [103] [104]. Este banco de pruebas incluye 15 test diferentes, computacionalmente muy costosos y que requieren tiempos de ejecución demasiado largos como para ser utilizados en la simulación de redes de sensores inalámbricas. Durante el proceso de diseño es preciso disponer rápidamente de los resultados de las simulaciones, con objeto de reducir el tiempo y coste de desarrollo. Es por esto que se ha definido una medida de la aleatoriedad computacionalmente más sencilla que los test estándar y que permite estimar la seguridad antes del despliegue de la red real.

En resumen, la nueva métrica permite identificar cuál es el nivel de seguridad que un método criptográfico provee a las transmisiones de datos en una red de sensores inalámbrica, al tiempo que:

1. Permite estimar, con un único valor, el nivel de seguridad de cualquier método criptográfico.

2. Consigue una mayor flexibilidad y granularidad que los test estándar, ya que evalúa la seguridad mediante un valor numérico. Los test estándar solo informan del número de pruebas correctamente ejecutadas (aproximación pasa/no-pasa) pero no cuantifican numéricamente el resultado de las mismas.

3. Reduce el tiempo de estimación, lo que posibilita su integración en el entorno de simulación de redes inalámbricas.

4.1. Evaluación de métodos criptográficos: Estado del arte

El estudio de las técnicas criptográficas se realiza en la sección 5.2.1. Esta sección se centra en el impacto de dichas técnicas y en la evaluación de su seguridad. Existen múltiples trabajos que analizan el impacto de la criptografía en distintos tipos de sistemas. En [105] se comparan las prestaciones de diferentes algoritmos criptográficos simétricos. En [106], los autores presentan resultados de experimentos realizados con AES y RC4, dos algoritmos de llave simétrica que comúnmente son usados en *WLANs*. Sin embargo, hoy en día, el uso de RC4 no esta tan extendido como el algoritmo Triple DES, por lo que la comparación no es completa.

En [52] se presenta un estudio de los problemas de seguridad existentes en redes de sensores inalámbricas. En primer lugar, se describen las limitaciones, los requisitos de seguridad y los ataques típicos (con sus correspondientes contramedidas) a este tipo de redes. A continuación, se presenta una visión general de los problemas de seguridad en WSN. Una de las conclusiones de este artículo es que la criptografía de llave simétrica es superior a los esquemas de llave asimétrica en cuanto a velocidad y consumo de energía. En [107] se evalúan algoritmos de criptografía integrales adecuados para redes de sensores inalámbricas. Para ello, se analizan distintos aspectos como por ejemplo, ciclos de

reloj, tamaño del código o consumo de energía. Sin embargo, dicho trabajo únicamente incluye algoritmos simétricos, como RC4 o AES.

Hasta donde ha estudiado el autor de esta tesis, en el estado del arte no existen propuestas para estimar el nivel de seguridad de los distintos algoritmos criptográficos que son utilizados en las redes de sensores inalámbricas. Sin embargo, se han propuesto y usado algunos test que pueden detectar cuando un algoritmo de encriptación puede considerarse seguro. Por ejemplo, la aleatoriedad de salida de un algoritmo criptográfico es una propiedad que se ha utilizado para evaluar algoritmos criptográficos [102]. El test estándar más utilizado es el "NIST *Statistical Test Suite*" [103] [104], que incluye 15 pruebas distintas. Dichas pruebas evalúan la aleatoriedad de una secuencia de salida mediante diferentes algoritmos. Aunque cada prueba proporciona una "métrica" (*P-value*), dicho valor no puede ser utilizado directamente para evaluar la seguridad. El "*P-value*" indica el nivel de significancia del test (o error de tipo I) el cual evalúa "la probabilidad de que el test indique que la secuencia no es aleatoria cuando realmente es aleatoria [102]". Para la evaluación de la seguridad, el error de tipo II [102] puede ser más relevante. Para una prueba, "el error de tipo II proporciona la probabilidad que el test indique que la secuencia es aleatoria cuando no lo es [102]". Sin embargo, el error de tipo II no es un valor fijo y es más difícil de calcular que el error de tipo I "debido a los múltiples tipos de no-aleatoriedad [101]". Debido a las limitaciones de los test y los "*P-values*", normalmente se utiliza directamente el número de test "pasados" para evaluar la aleatoriedad.

Un test de aleatoriedad se puede describir como una prueba de hipótesis estadística en la que la hipótesis probada es la aleatoriedad de los datos generados. Una de las pruebas de hipótesis más populares es la prueba de chi-cuadrado, la cual es frecuentemente utilizada en análisis estadístico. Algunas pruebas de chi-cuadrado adaptativo se han propuesto como pruebas de aleatoriedad [108] pero requieren cálculos demasiado complejos y heurísticas específicas. En esta sección se va a proponer una nueva métrica que modifica el test de chi-cuadrado para reducir la complejidad de los cálculos y mejorar los resultados. En la sección 4.2 se presenta la definición de la métrica y como se usa para evaluar los métodos criptográficos.

4.2. Métrica de estimación de seguridad: SEM (*Security Estimation Metric*)

En esta sección se va a presentar una métrica que mide el nivel de seguridad que un algoritmo criptográfico proporciona. Dicha métrica permitirá seleccionar durante el proceso de diseño la estrategia de encriptación que proporcione el mejor balance entre seguridad, consumo y tiempo de ejecución. El objetivo de la métrica es evaluar la aleatoriedad de la salida encriptada, ya que las técnicas de encriptación más seguras generan códigos encriptados más aleatorios.

Muchas métricas han sido utilizadas para medir la aleatoriedad de una señal, desde la entropía hasta funciones estadísticas clásicas como chi-cuadrado. La entropía se puede definir como una medida de la incertidumbre de una fuente de información. En el ámbito de la teoría de la información, la entropía también puede ser definida como la cantidad de información promedio que contienen los símbolos usados. En otras palabras, la entropía puede ser interpretada como el grado de desorden o aleatoriedad de un sistema. En el campo de la teoría de la información, la entropía se evalúa con la ecuación de Shannon.

Sin embargo, la función clásica de entropía puede ser menos sensible que otras funciones estadísticas para ciertas aplicaciones [109]. Por esta razón, el conjunto de test estándar de NIST [103] no está limitado a medir la entropía clásica (test de entropía aproximada -*approximate entropy test*-) e incluye otras 14 métricas como la frecuencia o la transformada discreta de Fourier, DFT [103]. Para definir la nueva métrica se han evaluado varias técnicas propuestas en publicaciones anteriores y se ha logrado identificar una nueva función que proporciona una precisa estimación de la aleatoriedad con un coste computacional bajo. La métrica propuesta se va a describir como una heurística en esta sección.

Una forma de evaluar la aleatoriedad de datos encriptados es analizar la función de distribución de los caracteres (bytes) cifrados. En el caso de que algún caracter aparezca con distinta frecuencia, se puede deducir que el nivel de seguridad es bajo y que el sistema podría ser vulnerable a ataques mediante métodos de

criptoanálisis. En la Figura 55 se puede apreciar la diferencia entre la distribución de valores de los bytes de un texto no cifrado y del mismo texto encriptado. Como muestra la figura, en el texto no cifrado (texto plano) hay fuertes variaciones de la frecuencia de aparición de caracteres.

Figura 55: Distribución de caracteres de un mensaje plano y encriptado

Partiendo de esta observación, la métrica de seguridad propuesta usa la desviación estándar de la frecuencia de ocurrencia de los caracteres encriptados. La desviación estándar muestra cuanta variación o dispersión existe al comparar un suceso con la medida esperada de un caso ideal, como es una secuencia completamente aleatoria. Una desviación estándar baja indica que los datos tienden a estar muy cerca de la medida esperada para una señal aleatoria. Por el contrario, una desviación estándar alta indica que los datos se extienden a lo largo de un amplio rango de valores por lo que la aleatoriedad del mensaje es baja. La ecuación no normalizada que evalúa la seguridad (SEM, *Security Estimation Metric*) se muestra en la ecuación 1.

$$SEM_{unnormalized}(N) = \sqrt{\frac{\sum_{i=1}^{N}(x_i - x_{mean})^2}{N}} \quad (1)$$

Ecuación 1: métrica de estimación de la seguridad (SEM) no normalizada

Donde N indica el número total de caracteres codificados (el tamaño del alfabeto), X_{mean} es la frecuencia media de ocurrencia de los caracteres encriptados en una secuencia aleatoria ideal y X_i es la frecuencia de ocurrencia del carácter

encriptado i en la secuencia analizada. Un valor alto de esta ecuación significa que la secuencia encriptada está lejos de ser una distribución aleatoria. En cambio un valor bajo indica que la secuencia podría ser aleatoria. Para obtener resultados precisos es recomendable realizar esta estimación con un gran número de muestras de mensajes. Sin embargo, debido a las limitaciones de memoria de los nodos de las redes inalámbricas y a la necesidad de ahorrar energía, los mensajes entre los nodos de la red suelen ser de pequeño tamaño. Como el valor obtenido de la ecuación (1) puede variar dependiendo del número de caracteres encriptados y del tamaño del alfabeto (N), se pueden obtener resultados contradictorios cuando se miden diferentes casos o tamaños de mensajes distintos. Esta limitación dificulta la evaluación del valor obtenido con la ecuación (1) para saber si se trata de una secuencia aleatoria. Con el fin de hacer frente a esta limitación, se propone una estrategia de normalización de la ecuación (1) que nos permite mejorar la identificación de la aleatoriedad. Para ello se propone comparar la secuencia encriptada con la secuencia aleatoria que se obtiene con un generador de caracteres aleatorio estándar (generador patrón). De esta forma, la métrica (SEM) se define con la siguiente ecuación normalizada:

$$SEM = \min_N\left(\frac{(SEM\_random_{unnormalized}(N))}{SEM_{unnormalized}(N)}\right) = \min_N\left(\sqrt{\frac{\sum_{i=1}^{N}(r_i - x_{mean})^2}{\sum_{i=1}^{N}(x_i - x_{mean})^2}}\right) \quad (2)$$

Ecuación 2: métrica de estimación de la seguridad (SEM)

Donde r_i es la frecuencia media de ocurrencia del carácter i en una secuencia aleatoria y los otros parámetros son similares a los identificados en la ecuación (1). La ecuación (2) evalúa la ecuación (1) con diferentes tamaños de alfabetos (típicamente 1, 2 y 3 bytes por caracter, por lo tanto N=28, 216, 224) y el valor obtenido será el mínimo de todos. Únicamente se han seleccionado tres tamaños distintos de alfabetos porque los experimentos realizados han demostrado que no hay ventajas en utilizar otros tamaños.

De acuerdo con la ecuación (2), la medida de seguridad se calcula como la raíz del cociente entre la estimación de la varianza de un caso aleatorio real (para un

alfabeto y longitud de secuencia específica) y la distribución de la varianza del mensaje a evaluar. Dependiendo de la aplicación, la medida “SEM_random” puede ser previamente calculada y guardada en memoria, lo que reduce el cómputo durante la estimación. En otros casos, esta medida puede ser estimada dinámicamente (por ejemplo, el mensaje aleatorio ideal puede ser creado con la función “rand” de una librería matemática).

Es interesante remarcar que el uso de una función aleatoria en vez de la definición matemática teórica del valor SEM_random permite a la métrica tener en cuenta las diferencias de varianza admisibles en una implementación real de un generador de números aleatorios, con diferentes alfabetos y tamaños de mensajes. Debido a la definición de la ecuación (2) se puede observar que la máxima seguridad se obtiene cuando SEM toma el valor “1”, mientras que cuando SEM tome un valor cercano a “0” la seguridad es mínima.

En resumen, la métrica SEM presentada en la ecuación (2) evalúa la desviación estándar normalizada de la frecuencia de ocurrencia de los caracteres de un mensaje. Gracias a esta métrica es posible indicar, con un único número, el nivel de seguridad de una secuencia de datos encriptados. Como ya se comentó al principio de esta sección, la métrica SEM se ha presentado como una heurística. Sin embargo, desde un punto de vista estadístico, la ecuación (2) puede ser vista como un test chi-cuadrado que compara una secuencia *aleatoria* ideal con una secuencia encriptada.

4.3. Evaluación de la métrica propuesta

Con el fin de demostrar las ventajas de la métrica SEM, se van a evaluar varios algoritmos criptográficos de uso frecuente. El análisis se centra en los algoritmos de encriptación simétricos AES (*Advanced Encryption Standard*) [110] y TDES (*Triple Data Encryption Algorithm*). Estos algoritmos simétricos se han seleccionado por ser de uso común en redes de sensores inalámbricas debido a que su implementación normalmente requiere menos energía que las técnicas asimétricas. AES es un eficiente método criptográfico estándar que usa la misma clave para

encriptar y desencriptar (clave simétrica). Con objeto de aumentar la seguridad, AES realiza un cierto número de rondas o iteraciones de un algoritmo básico.

El número de rondas y los bits de la clave determinan la robustez (o nivel de seguridad) de la encriptación. En este apartado, para evaluar la métrica propuesta (SEM) se van a utilizar implementaciones de AES con diferentes números de rondas (específicamente 6, 10 y 14 rondas) además de TDES.

Para ampliar la comparación, también se evaluará una encriptación simple que muestra un método de baja seguridad. Esta encriptación ligera se realiza redistribuyendo los símbolos para una secuencia específica. El seudocódigo de esta encriptación ligera se presenta en la Figura 56.

```
For (i=0 to encrypted block size)
     EncryptedText[i] = PlainText[i] + i * mod(256)
```

Figura 56: Seudocódigo de la encriptación ligera

En el análisis se incluye también la distribución de bytes de un generador de números aleatorio estándar. Esta distribución se denominará “caso ideal” en el resto de la sección ya que, como se ha comentado anteriormente, los modelos de distribución aleatorios representan una secuencia con seguridad ideal, con máxima aleatoriedad.

Para validar la métrica de seguridad, se han identificado 7 niveles de seguridad. Estos niveles están ordenados de menos (nivel 0) a más seguridad (nivel 7). Estos niveles de seguridad se han definido teniendo en cuenta comparaciones entre los distintos algoritmos criptográficos. Los niveles de seguridad identificados en esta sección son:

1. Sin encriptación.
2. Encriptación ligera.
3. Encriptación TDES.
4. Encriptación AES-128 con 6 rondas (clave con 128 bits).
5. Encriptación AES-192 con 10 rondas (clave con 192 bits).

6. Encriptación AES-256 con 14 rondas (clave con 256 bits).
7. Caso ideal (secuencia aleatoria).

4.4. Comparación grafica entre diferentes estrategias criptográficas

Esta sección presenta una comparación gráfica de las funciones de distribución de los mensajes encriptados. Esta comparación gráfica permite justificar la métrica seleccionada. Para ello, vamos a asumir que tenemos un texto sin cifrar que contiene 256 diferentes posibles caracteres (valores de bytes). El tamaño del texto es de 2 Mbyte.

Como se observa en la Figura 57.a, cuando el texto no está encriptado, algunos bytes tienen mucha más frecuencia de aparición que otros. En el otro extremo está el caso ideal (Figura 57.e) con datos completamente aleatorios. Merece la pena mencionar que la secuencia aleatoria ha sido generada con la función “rand” de la librería matemática estándar de C. En este último caso, no hay una frecuencia de ocurrencia de los bytes que sea claramente superior o inferior que las otras, por lo que es más difícil detectar el mensaje encriptado. Entre estos dos casos se muestran la encriptación ligera (Figura 57.b), TDES (Figura 57.c) y AES (Figura 57.d). En la gráfica de la encriptación ligera se puede apreciar que su distribución no es tan uniforme como el caso de TDES o AES. Pero es más segura que la no encriptada (Figura 57.a). En el caso del TDES y el AES, tienen una frecuencia de ocurrencia mucho más similar al caso ideal, pero es difícil decidir con esta información gráfica cuál de ellos es más seguro.

Revisando la Figura 57 se puede entender visualmente que los mensajes encriptados con algoritmos seguros (Figura 57.c y Figura 57.d) son más difíciles de piratear que el texto plano o la encriptación ligera (Figura 57.a y Figura 57.b). Sin embargo, los resultados mostrados en la Figura 57 no son medidas prácticas para medir el nivel de seguridad obtenido por un algoritmo criptográfico. Por ello, es

necesario utilizar la métrica SEM descrita en la Ecuación 2 para medir la seguridad de las comunicaciones en las redes de sensores inalámbricas.

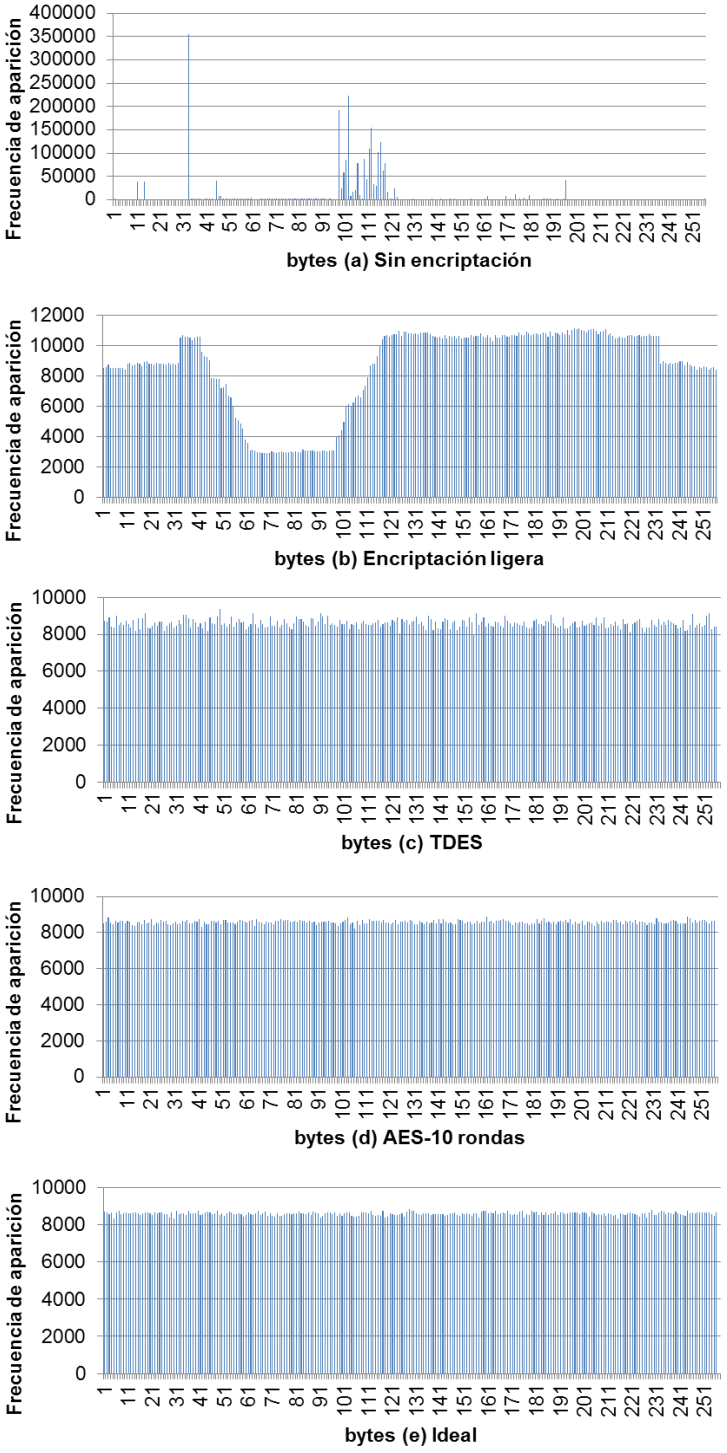


Figura 57: Frecuencia de ocurrencia de los bytes para 5 casos de encriptación

4.5. Comparación de la métrica SEM y los test estándar de NIST

Con el fin de validar la métrica de seguridad, se van a comparar sus resultados con los proporcionados por los test estándar propuestos por NIST [103] [104]. Para mejorar la fiabilidad, el estudio incluye un número muy elevado de mensajes de diferentes tipos. Por esta razón se han incluido más de 6000 mensajes (o ficheros) distintos con diferentes formatos: textos en inglés (formato txt), imágenes en formato comprimido (formato jpeg) e imágenes en formato plano y sin compresión (formato pgm). Los mensajes han sido encriptados utilizando los 7 niveles de seguridad que se propusieron en la Sección 4.3. En la Tabla 16 se presentan los resultados de un subconjunto de 4 de los test propuestos por NIST cuando son aplicados a mensajes que incluyen un texto en inglés. En concreto, se presentan los *P-values* del test 1 (Frecuencias), test 3 (*Runs*), test 6 (Transformada Discreta de Fourier) y test 12 (Entropía aproximada). Desde el punto de vista de los test de NIST, el *P-value* representa la significancia o probabilidad de aceptación de cada test: si el *P-value* es mayor que 0.01, el test tendrá un resultado positivo o "test pasado" [103]. Es importante observar las diferencias entre P-*values* para los mismos niveles de seguridad. Por lo tanto se puede concluir que la correlación entre los *P-values* y los niveles de seguridad es pobre. Por ejemplo, el *P-value* estimado para el test 1 de TDES es mayor que el del AES-256. Si analizamos la literatura, está demostrado que AES es más seguro que el TDES, lo que contradice los valores de los P-*values*. Debido a esta mala correlación entre los *P-values*, la métrica de seguridad de los test de NIST se ha obtenido teniendo en cuenta el número de test pasados. En las siguientes tablas se podrá comprobar que esta medida (número de test pasados) proporciona una mejor correlación con los niveles de seguridad que un único test en particular. Adicionalmente, la Tabla 16 demuestra que la métrica propuesta SEM tiene una mejor correlación con los niveles de seguridad que otros test, como pueden ser el obtenido con la Transformada Discreta de Fourier (test 6) o la Entropía aproximada (test 12).

Tabla 16: *P-value* para 4 test de NIST (1, 3, 6 y 12) y la métrica SEM

		Frequency (test 1)	Runs (test 3)	DFT (test 6)	Approx. Entropy (test 12)	SEM
Menos seguro	**Sin encriptación**	0	0	0	0	0.0034
↓	**Encriptación ligera**	0	0	0	0	0.087
↓	**TDES**	0.74	0.73	0.53	0	0.744
↓	**AES-128**	0.53	0.911	0.534	0.122	0.996
↓	**AES-192**	0.74	0.73	0.213	0.017	0.997
↓	**AES-256**	0.35	0.73	0.91	0.35	0.999
Más seguro	**Caso ideal**	0.53	0.017	0.213	0.74	1

La Tabla 17, Tabla 18 y Tabla 19 comparan la métrica de seguridad propuesta (SEM) con el conjunto de test de NIST. En la Tabla 17 se presenta una comparación entre estas métricas para el caso de un mensaje con formato texto (fichero ".txt"). En la Tabla 18 se presentan los mismos datos pero para el caso de imágenes sin compresión (imágenes con formato ".pgm"). Por último, en la Tabla 19 se muestran los resultados para imágenes comprimidas (imágenes ".jpeg"). En la columna de los test de NIST se muestra la relación entre el número de test "pasados" y el número total de test. Por lo tanto, la métrica representada es el porcentaje de test pasados. La columna SEM representa el resultado de la métrica propuesta (Ecuación 2).

Tabla 17: Test de NIST y SEM para fichero de texto

		Test de NIST	SEM
Menos seguro	**Sin encriptación**	1/15=0.066	0.0034
↓	**Encriptación ligera**	2/15=0.133	0.087
↓	**TDES**	11/15=0.733	0.744
↓	**AES-128**	15/15=1	0.996
↓	**AES-192**	15/15=1	0.997
↓	**AES-256**	15/15=1	0.999
Más seguro	**Caso ideal**	15/15=1	1

Tabla 18: Test de NIST y SEM para imágenes pgm sin compresión

		Test de NIST	SEM
Menos seguro	**Sin encriptación**	1/15=0.066	0.0128
↓	**Encriptación ligera**	3/15=0.200	0.287
↓	**TDES**	11/15=0.733	0.836
↓	**AES-128**	14/15=0.933	0.947
↓	**AES-192**	15/15=1	0.956
↓	**AES-256**	15/15=1	0.975
Más seguro	**Caso ideal**	15/15=1	1

Tabla 19: Test de NIST y SEM para archivos jpeg con compresión

		Test de NIST	SEM
Menos seguro	**Sin encriptación**	2/15=0.133	0.004
↓	**Encriptación ligera**	13/15=0.866	0.852
↓	**TDES**	13/15=0. 866	0.923
↓	**AES-128**	15/15=1	0.944
↓	**AES-192**	15/15=1	0.997
↓	**AES-256**	15/15=1	0.977
Más seguro	**Caso ideal**	15/15=1	1

Para poder apreciar más fácilmente la correlación entre la métrica propuesta y los test de NIST, en la Figura 58 se representan gráficamente la correlación entre los resultados de la Tabla 17, la Tabla 18 y la Tabla 19. Se puede apreciar que la correlación entre ambas métricas es bastante buena.

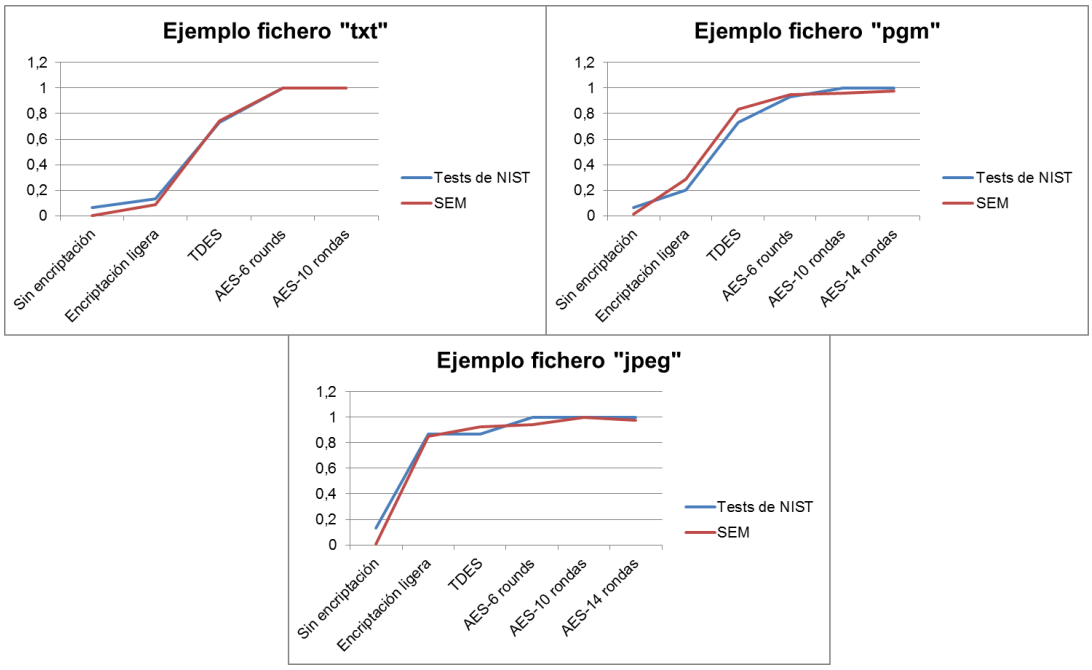


Figura 58: Ejemplos de los valores de seguridad obtenidos con ambas métricas

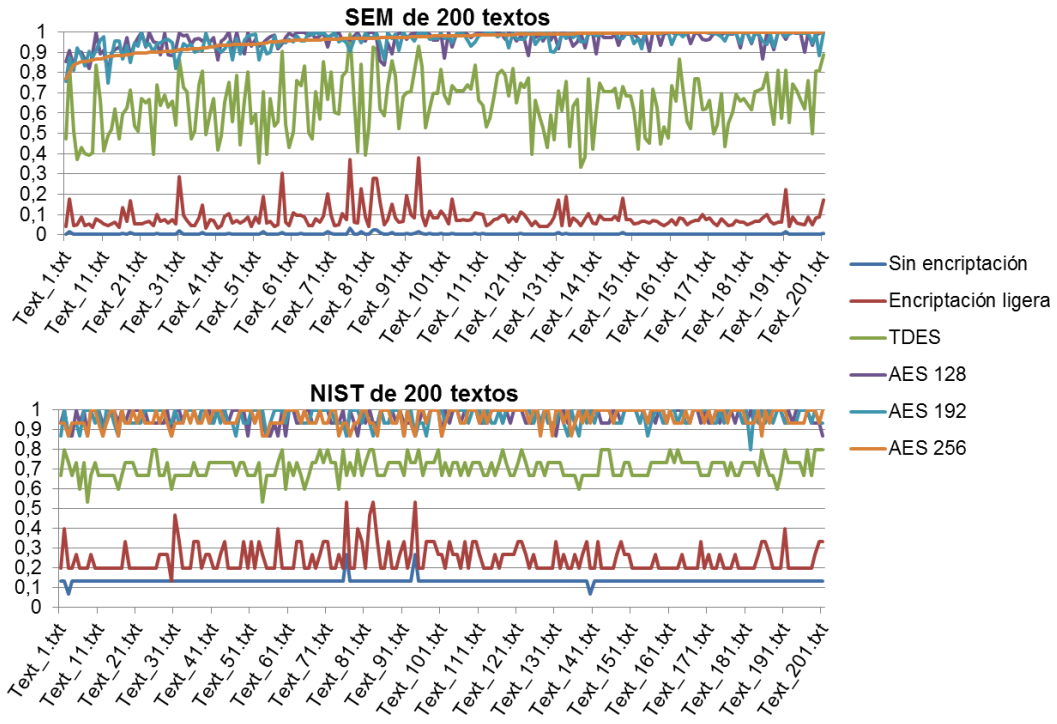


Figura 59: Valores de seguridad obtenidos para 200 textos

En la Figura 59 se puede ver una comparación entre las dos métricas (SEM y NIST) con 200 archivos de texto. Las gráficas están ordenadas según los valores

obtenidos por la métrica SEM para el caso de AES-256 (línea naranja). Como se puede observar, los resultados obtenidos por la métrica SEM (Figura 59 a) son muy similares a los obtenidos por la métrica de NIST (Figura 59 b). En el caso del texto encriptado con AES-256, el rango de la métrica SEM abarca desde 0.76 a 1 con una media de 0.96. En el caso de los test de NIST el rango abarca desde 0.87 a 1 con un valor medio de 0.96. Estos valores (máximo, mínimo y media) representados en la Figura 59 se pueden observar en la Tabla 20.

Tabla 20: Resultados de NIST y SEM para los textos

	Métrica propuesta SEM						**NIST Statistical Test Suite**					
	Sin encript.	Encript. Ligera	TDES	AES 128	AES 192	AES 256	Sin encript.	Encript Ligera	TDES	AES 128	AES 192	AES 256
Ave	0.004	0.08	0.64	0.96	0.97	0.97	0.13	0.24	0.70	0.96	0.96	0.96
Max	0.03	0.38	0.98	1	1	1	0.26	0.53	0.8	1	1	1
Min	0.002	0.03	0.33	0.79	0.74	0.77	0.06	0.13	0.53	0.86	0.80	0.87

En la Figura 60 se puede ver la comparación de las métricas NIST y SEM con 250 ejemplos de imágenes sin compresión (imágenes con formato pgm). Igual que en la Figura 59, los resultados están ordenados según los valores obtenidos por la métrica SEM para el caso de AES-256 (línea naranja). En estos ejemplos, el tamaño de los ficheros sin encriptar es más pequeño que en los otros ejemplos (alrededor de 75Kb) y, debido a ello, el rango entre las medidas obtenidas con NIST y la métrica SEM es mayor que en el caso anterior. Por ejemplo, en el caso de los ficheros encriptados con un AES-256, el rango generado por los test de NIST abarca desde 0.73 a 1, con un valor medio de 0.91. En cambio, con la métrica SEM, los valores van desde 0.69 a 1, con una media de 0.92. En la Tabla 21 están presentados los valores medios, mínimos y máximos representados en la Figura 60.

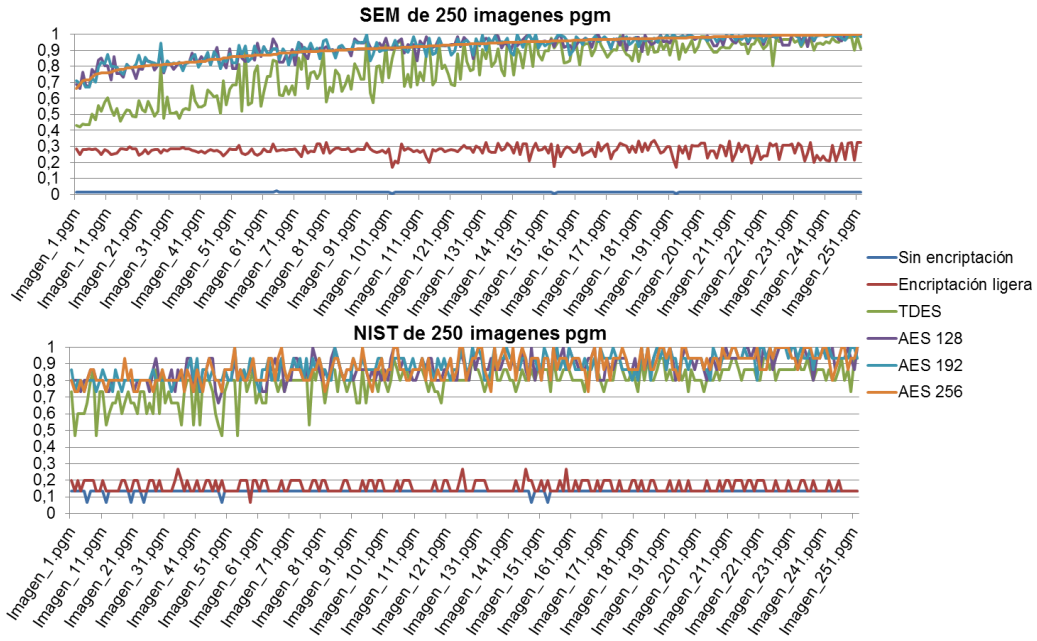


Figura 60: Valores de seguridad obtenidos para 250 imágenes pgm

Tabla 21: Resultados de NIST y SEM para las imágenes pgm

	Métrica propuesta SEM						NIST Statistical Test Suite					
	Sin encript.	Encript. Ligera	TDES	AES 128	AES 192	AES 256	Sin encript.	Encript. Ligera	TDES	AES 128	AES 192	AES 256
Ave	0.01	0.27	0.79	0.92	0.92	0.92	0.13	0.16	0.79	0.88	0.89	0.91
Max	0.02	0.34	0.97	1	1	1	0.13	0.27	0.93	1	1	1
Min	0	0.17	0.44	0.66	0.67	0.69	0.07	0.07	0.47	0.67	0.73	0.73

La Figura 61 es similar a la Figura 59 y Figura 60, pero para imágenes comprimidas con formato JPEG. En este caso se presentan los resultados obtenidos en cada nivel de seguridad después de evaluar 250 imágenes diferentes. Estos resultados son especiales debido a que el fichero original (sin encriptar) esta comprimido por el algoritmo JPEG. Según la literatura, una compresión podría comportarse como una encriptación, produciendo una cierta aleatorización de los datos. Por esta razón, los resultados de la encriptación ligera, por ejemplo, son más altos que en los casos presentados previamente. En general se observa que ambas métricas (SEM y NIST) incrementan sus valores, pero conservando la proporcionalidad. Como se muestra en la Tabla 22, el valor medio para la

encriptación TDES es de 0.92 en ambas métricas. Este valor es muy alto en comparación con los resultados presentados en la Tabla 20 y Tabla 21. Esta característica se puede apreciar también en el caso de la encriptación ligera.

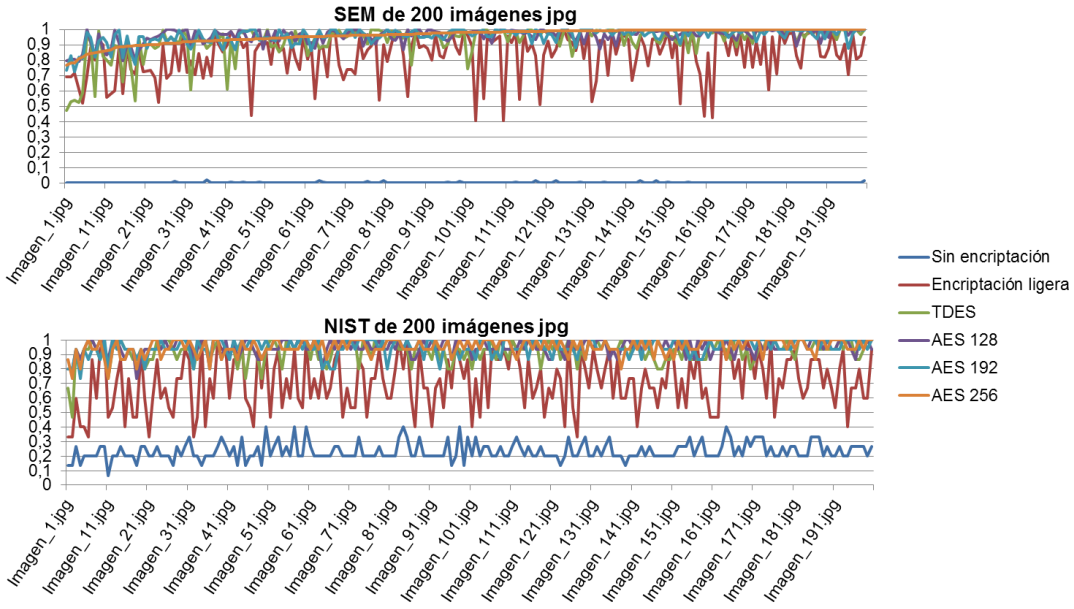


Figura 61: Valores de seguridad obtenidos para 200 imágenes jpeg

Tabla 22: Resultados de NIST y SEM para las imágenes jpeg

	Métrica propuesta SEM						**NIST Statistical Test Suite**					
	Sin encript.	Encript. Ligera	TDES	AES 128	AES 192	AES 256	Sin encript.	Encript. Ligera	TDES	AES 128	AES 192	AES 256
Ave	0.003	0.83	0.92	0.96	0.96	0.97	0.23	0.71	0.92	0.94	0.95	0.96
Max	0.02	0.99	0.99	1	1	1	0.4	1	1	1	1	1
Min	0.002	0.40	0.47	0.79	0.72	0.77	0.07	0.33	0.66	0.73	0.73	0.73

Como se puede observar en las Tablas y las Figuras presentadas en esta Sección, AES es el algoritmo de encriptación más seguro para las dos métricas estudiadas: SEM y test de NIST. Este resultado confirma los abundantes estudios previos que corroboran esta conclusión. También es bien sabido que TDES es menos seguro que AES. Esto se puede confirmar con la métrica propuesta ya que

los valores estimados con la métrica SEM en los casos encriptados con TDES son menores que los valores obtenido en los casos encriptados con AES. También se puede apreciar que la encriptación ligera muestra una seguridad mucho más pobre. Finalmente, para el caso de que no haya encriptación, el valor obtenido con SEM es claramente inferior (dos órdenes de magnitud en todos los casos) que cualquiera de los otros métodos criptográficos. Adicionalmente, el método de encriptación más seguro (AES) presenta siempre un valor de SEM muy cercano al valor ideal. Por lo tanto, se puede observar que hay una buena correlación entre los valores obtenidos por la métrica SEM y las comparaciones entre diferentes métodos criptográficos del estado del arte.

Finalmente, la Tabla 23 presenta los tiempos de ejecución de ambas métricas. Se puede observar que la métrica propuesta es mucho más rápida que los test de NIST (más de dos órdenes de magnitud). Esta reducción del tiempo de ejecución es muy importante en el ámbito de la simulación de redes de sensores inalámbricas, ya que para guiar el proceso de diseño se necesita ejecutar un gran número de estimaciones de seguridad, consumo y tiempo de ejecución. En la Tabla 23 se presenta el tiempo de ejecución total que se requiere para calcular la métrica SEM, así como el número de test de NIST pasados para un texto de 1 Mbyte. En el caso de la métrica SEM únicamente se requieren 0.009 segundos mientras que la métrica de NIST requiere 5.1 segundos para procesar 1 secuencia. Sin embargo, como se comenta en [103], para poder proveer resultados estadísticamente significativos con los test de NIST es necesario procesar al menos 55 secuencias. En este caso, el tiempo de ejecución de los test de NIST aumenta por encima de los 5 minutos (313 segundos). Estos tiempos de ejecución se han obtenido con un ordenador Intel i5 con 4 núcleos, con una frecuencia de reloj de 3.2 GHz y 4 GB de RAM.

Tabla 23: Tiempos de ejecución

	SEM	**Test de NIST**	
		1 Secuencia	**55 Secuencias**
Tiempo medio de ejecución	0.009 s	5.8 s	313.53

5. Firmware seguro y eficiente

El desarrollo de redes de sensores inalámbricas seguras no solo requiere que la funcionalidad implementada en los nodos de la red soporte ataques, sino que también es necesario disponer de implementaciones seguras y eficaces de ciertos servicios de la plataforma software que son independientes de la funcionalidad del nodo. Durante la realización de esta tesis, se detectó la existencia de dos problemas que afectan a la seguridad y eficiencia de las redes de sensores inalámbricas.

El primer gran problema es la actualización de estas redes una vez desplegadas. Esta tarea conlleva un consumo de energía que puede afectar a la duración de la batería de los nodos y, por ello, al tiempo de vida de la red. Para paliar este problema se ha desarrollado un método de actualización parcial del firmware del nodo.

El segundo gran problema es garantizar que cuando el nodo inicia su actividad lo hace con el programa correcto y no con un código malicioso. El momento del arranque del nodo (*booting*) es un punto crítico para la seguridad que ha sido ampliamente estudiado en la literatura de los sistemas tradicionales. En las redes de sensores este proceso es aún más crítico, al ser dichas redes normalmente accesibles para los atacantes y no disponer de vigilancia. Es por ello que en esta sección se presenta un método de arranque que permite seleccionar la opción que mejor combina la seguridad con el consumo de energía del nodo.

5.1. Actualización Parcial de firmware

La capacidad de modificar la programación de los nodos de las redes inalámbricas una vez desplegadas es un requisito esencial de este tipo de sistemas. Estas redes normalmente están diseñadas para funcionar durante largos periodos de tiempo, en los cuales puede ser necesario modificar el software de los nodos para añadir nuevas funcionalidades o corregir errores. Añadir la posibilidad de actualización del firmware de manera inalámbrica (OTAP, *Over The Air Programming*) facilita las tareas de mantenimiento, ya que muchas veces estas redes están desplegadas en entornos con difícil acceso a los nodos. Hay que tener en cuenta que, en los sistemas convencionales, el mantenimiento del software representa entre el 60% y el 70% de su coste [111]. De este esfuerzo, alrededor del 50% es perfectivo (mejora de la funcionalidad), el 21% es correctivo, el 24% es adaptativo (soporte a nuevas plataformas) y el ´último 4% representaría el esfuerzo preventivo (auto verificación) [112]. Las modificaciones de software se realizan normalmente mediante adaptaciones (*wrappers*) y parches [113].

La programación inalámbrica implica la transmisión por la red de las modificaciones de software, su almacenamiento e instalación en el nodo. Como las redes de sensores inalámbricas tienen unas restricciones de consumo muy importantes es esencial reducir el consumo de energía durante dicho proceso. Por lo tanto, la cantidad de información que se transmite por la red durante la actualización de firmware debe reducirse al mínimo, ya que el consumo de esta tarea está directamente ligado con la cantidad de paquetes de radio que se envían. Además, cuanto mayor es el tamaño de la actualización, la posibilidad de error en la transmisión aumenta, lo que incrementa el número de paquetes reenviados. Si se consigue reducir el tamaño de la actualización, se podrá conservar por más tiempo la batería de los nodos y ampliar la vida de la red.

Se han identificado dos enfoques principales con los que afrontar la reducción del número de paquetes enviados durante la actualización. La primera aproximación intenta optimizar el protocolo de diseminación de la actualización. Cuanto más eficiente sea el protocolo, menor será el número de transmisiones

necesarias para que la actualización llegue a todos los nodos. La segunda aproximación busca reducir el tamaño de la actualización. Esta aproximación será la utilizada en esta tesis.

Se han propuesto varias estrategias para actualizar el firmware en campo. La manera más simple y utilizada consiste en cargar el firmware completo, de manera que sustituta a todo el software que se encontraba en el nodo en la versión anterior. Esto supone que para cualquier cambio, por mínimo que sea, es necesario enviar el firmware completo. Por esta razón, el principal inconveniente de esta aproximación es que el consumo de energía y el tiempo de actualización son muy altos, ya que es necesaria la transmisión de todo el firmware en cada actualización. Otras estrategias (como la presentada en esta sección) intentan enviar solo la parte del firmware que se ha modificado (actualización incremental o parcial).

Además de reducir la cantidad de información a transmitir durante la actualización, es necesario tener en cuenta que el firmware se almacena en el nodo en una memoria de tipo *flash*. Como se verá en el próximo apartado, para alargar la vida útil de la memoria flash es necesario reducir el número de veces que se borra dicha memoria, acción que puede ser necesaria cada vez que se modifica el firmware.

En esta sección se propone una técnica de actualización parcial (incremental) que, además de transmitir solo la parte de firmware que se ha modificado, intenta reducir el número de operaciones de borrado de la memoria *flash*. Dicha técnica puede ser usada en cualquier dispositivo y, a diferencia de otras aproximaciones, no necesita utilizar la Unidad de Gestión de Memoria, MMU (*Memory Management Unit*). Como la mayoría de los procesadores de sistemas embebidos no disponen de Unidad de Gestión de Memoria (MMU), la técnica desarrollada está especialmente dirigida a sistemas embebidos, como las redes de sensores inalámbricas. Esta técnica ha dado lugar a la patente presentada en [114].

5.1.1. Actualización de firmware: Estado del arte

En la literatura existen múltiples técnicas de actualización de firmware para redes de sensores inalámbricas que abarcan desde técnicas sencillas, como puede ser el reemplazo de la imagen completa, a soluciones más complejas, como actualizaciones incrementales o parciales. En [115], [116] y [117] los autores presentan diferentes protocolos para actualizar la imagen completa del firmware. Estas técnicas reemplazan toda la imagen (incluyendo aplicaciones, sistema operativo y *drivers*) lo que requiere, en la mayoría de los casos, mucho tiempo y un alto consumo de energía. Para solventar estos problemas, algunos autores proponen técnicas de reprogramación incremental [118] y [119]. Dichas aproximaciones buscan reducir el tamaño de las actualizaciones, lo que reduce el consumo y aumenta el tiempo de vida del nodo. Las actualizaciones incrementales buscan poder transmitir únicamente las modificaciones existentes entre la versión residente en el nodo y la nueva versión de firmware, para evitar el envío de código duplicado. Como se puede apreciar en la Figura 62, las dos versiones del firmware son comparadas para extraer la secuencia de cambios a realizar. En ella únicamente están incluidas las modificaciones del código, lo que permite reducir su tamaño al mínimo.

Figura 62: Reducción del tamaño de la actualización buscando diferencias

Este esquema básico permite reducir el número de paquetes que será necesario transmitir por la red durante el proceso de actualización. Dentro de las técnicas que intentan reducir el tamaño de la actualización se pueden identificar dos aproximaciones. La primera aproximación asume que el firmware se ha desarrollado de forma modular, de tal modo que únicamente sea necesario transmitir los módulos que se han modificado, en lugar del firmware completo. La segunda aproximación utiliza actualizaciones incrementales, donde únicamente se transmiten los cambios respecto al software que se está ejecutando en el nodo en el momento de la actualización.

Un ejemplo de estas técnicas se presenta en el documento [119], donde los autores utilizan una técnica de actualización incremental que tiene en cuenta las diferencias entre el firmware antiguo y el nuevo. Las actualizaciones son realizadas mediante una secuencia de comandos (normalmente conocida como "*script*") con las diferencias entre las versiones de firmware. Dicha secuencia es la que se transmite a través de la red de comunicaciones. El problema de este tipo de técnicas es que el tamaño de los *scripts* de actualización no es necesariamente congruente con la extensión real de la actualización. En la mayoría de los casos, un pequeño cambio en el código fuente da como resultado una imagen del firmware muy similar pero con las distintas direcciones de las memorias o con el código desplazado. La Figura 63 muestra gráficamente este problema. En ella se pueden observar las diferencias entre el mapa de memoria del firmware original y el firmware nuevo. Vemos como la actualización del nuevo firmware ha modificado la Función 2, dando como resultado que ésta ocupe un tamaño mayor en la memoria que en la versión de firmware anterior. Esto ha ocasionado que las direcciones de los objetos o las funciones que están posicionadas después de la Función 2 hayan cambiado debido al desplazamiento (Función 3 y Función 4). Por lo tanto, cuando el código necesite ejecutar la Función 3, será necesario llamar o saltar a su nueva dirección, ya que la antigua ahora es parte del código de la Función 2. Esto conlleva actualizar la dirección de cada llamada a la Función 3 dentro del código. También se puede apreciar que habrá que actualizar todo el código que está a continuación de la Función 2, ya que se ha visto desplazado. Como resultado, se obtiene un *script* de actualización con un tamaño mucho mayor del que inicialmente se podría

esperar. Este problema no está limitado a las funciones, sino a todos los objetos en los cuales la posición de la memoria puede cambiar como, por ejemplo, los datos constantes almacenados en *flash* y las variables estáticas o globales almacenadas en RAM.

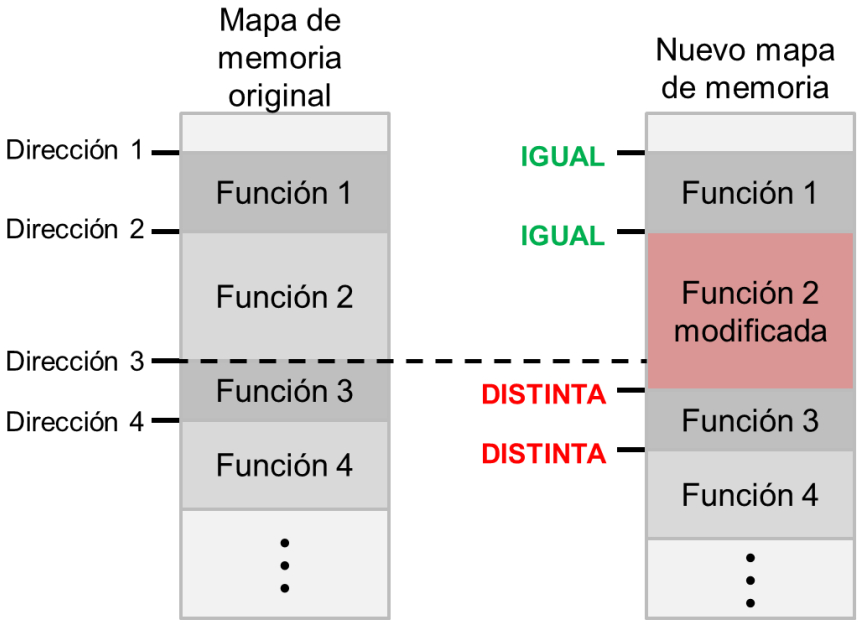


Figura 63: Diferencias en las direcciones de memoria después de actualizar

Con el diseño de memoria típico mostrado en la Figura 63, donde las funciones están ordenadas consecutivamente, es muy difícil obtener actualizaciones eficientes que requieran pocos cambios.

Una solución a este problema se analiza en [120], donde los autores proponen la introducción de un "espacio de reserva" o "*slop space*" entre las funciones del firmware, de forma que permitan a la función aumentar su tamaño sin que se modifiquen las demás funciones (Figura 64). Esto permite almacenar la nueva función en las posiciones de memoria de la función inicial, sin desplazar las posiciones de las otras funciones. Sin embargo, aunque esta solución pudiese evitar el problema en ciertos casos (actualizaciones en las que las funciones no crecen más que el espacio libre), los problemas surgen cuando la modificación de la función requiere un espacio que es mayor que su "*slop space*". Esto obliga a desplazar al resto de las funciones con el consiguiente incremento del tamaño de la

actualización. Además, la cantidad de memoria *flash* usada crece y se usa ineficientemente al dejar muchos espacios sin utilizar.

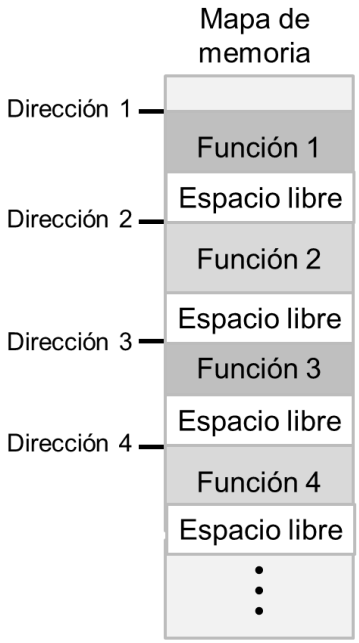


Figura 64: Esquema dejando espacio libre entre funciones

Otras técnicas de actualización se basan en el uso de máquinas virtuales (como por ejemplo [121], [122], [123] y [124]). Estas técnicas proponen el uso de la tecnología de virtualización y su principal inconveniente es su complejidad. El consumo de potencia durante la compilación en tiempo de ejecución, el *linkado* y el proceso de ejecución es mucho mayor que si se usa código nativo, por lo que el sobrecoste en términos de uso de energía es prohibitivo. Por ejemplo, en [123] se presenta Mate, una máquina virtual compacta diseñada específicamente para redes de sensores inalámbricas. Esta máquina virtual incluye soporte para actualización de código, pero el incremento en el consumo de energía es muy grande. Además, esta herramienta únicamente soporta actualizaciones de las aplicaciones pero, en muchos casos, también es necesario modificar el resto del firmware (por ejemplo, modificar el sistema operativo).

Existen otras técnicas basadas en uso de librerías dinámicas o *linkado* dinámico. En ellas se cargan dinámicamente distintos módulos en el dispositivo, usando código independiente de la posición (PIC, *Position Independent Code*). Por ejemplo,

en [125] se presenta una técnica que permite cargar dinámicamente módulos individuales en un nodo de la red. Dichos módulos se comunican y ejecutan mediante mensajes y tablas.

Otros trabajos combinan varias técnicas. Por ejemplo, [126] combina el uso de llamadas indirectas a funciones (código independiente de la posición) con un *script* que incluye las diferencias entre versiones de firmware. Uno de los inconvenientes de esta aproximación es que, aunque el código sea independiente de la posición, no se solucionan las dependencias del árbol de llamadas. Por ello, si una función llama a otra función que haya cambiado de posición debido a una actualización, es necesario modificar la dirección de la llamada de la primera.

Las técnicas incrementales comentadas anteriormente requieren el borrado y la reescritura completa de la memoria en donde se almacena el firmware cada vez que haya una actualización. Para comprender el impacto de la reescritura de la memoria es preciso analizar la arquitectura hardware del sistema.

Los nodos de la red son dispositivos electrónicos que integran unidades de almacenamiento de datos (memorias) para guardar el firmware. Tradicionalmente, el software embebido se almacena en módulos de memoria ROM (*Read-Only Memory*, Memorias de Solo Lectura) o *flash*. Sin embargo, la plataforma hardware normalmente integra otros tipos de unidades de almacenamiento, que suelen ser clasificadas en dos categorías: memorias volátiles y no-volátiles. Las memorias volátiles, como las RAM (*Random Access Memory*, Memorias de Acceso Aleatorio), se borran si hay una interrupción del suministro de energía. Estas memorias son baratas y rápidas, pero deben estar continuamente alimentadas para no perder la información que almacenan, lo que hace que consuman mucha energía. Las memorias no volátiles, como la ROM, conservan la información aún sin suministro de energía. Entre las memorias no volátiles destaca la memoria *flash*, en la que bloques de datos pueden ser eléctricamente borrados y reprogramados. Como la programación de un sistema embebido no se debe borrar si el suministro de energía se interrumpe, normalmente el firmware del sistema se almacena en memorias de tipo *flash*. Las memorias *flash*, al contrario de las RAM, pueden preservar los datos largo tiempo sin necesidad de mantener el suministro de

energía, pero son más caras, más lentas y requieren una interfaz de operación más compleja. Antes de poder escribir en la memoria *flash*, es necesario borrar el bloque de memoria o banco en el que se va a escribir. El borrado de un bloque es un proceso que requiere no solo tiempo sino también una cierta cantidad de energía. Cuando se borra un bloque solo es posible escribir una vez en una posición de memoria. Si se desea reescribir una posición de memoria, es necesario borrar todo el bloque de memoria otra vez y reescribir las posiciones del bloque que no se deseen modificar. Además, la necesidad de borrado previo reduce la vida útil de las memorias no volátiles, cuyos ciclos de borrado están limitados (el fabricante no garantiza que el dispositivo funcione después de un cierto número de borrados).

La mayoría de las aproximaciones de actualización incremental parten del firmware actual (versión antigua) y con el *script* de actualización generan un firmware nuevo en la memoria RAM del dispositivo. A continuación, lo transfieren a la memoria *flash*, borrando previamente el firmware anterior. Debido a esto, es necesario el borrado completo de la memoria *flash*, aun cuando los cambios en el firmware sean mínimos. Como se ha comentado anteriormente, este proceso no solo requiere un tiempo y consumo de energía elevados, sino que también reduce la vida útil de la *flash*, lo que constituye un serio inconveniente que no es resuelto en la mayoría de las técnicas de actualización de firmware publicadas.

A continuación, se presenta una técnica que supera muchas de las limitaciones anteriormente comentadas. Dicha técnica de actualización parcial o incremental reduce el tráfico en la red (solo es necesario enviar las funciones modificadas) y el número de borrados de los bloques de la *flash*.

5.1.2. Técnica de actualización incremental propuesta

La técnica de actualización parcial propuesta en este capítulo mejora las aproximaciones clásicas de actualización incremental presentadas en el apartado anterior de estudio del estado del arte. Para poder resolver los problemas de dichas

técnicas clásicas como, por ejemplo, el problema del desplazamiento de las funciones actualizadas o el borrado de *flash*, se propone almacenar las funciones actualizadas a continuación del firmware antiguo, en una región libre de la memoria *flash* del nodo. Con esta solución conseguiremos que las funciones antiguas no se modifiquen y mantengan su posición en memoria.

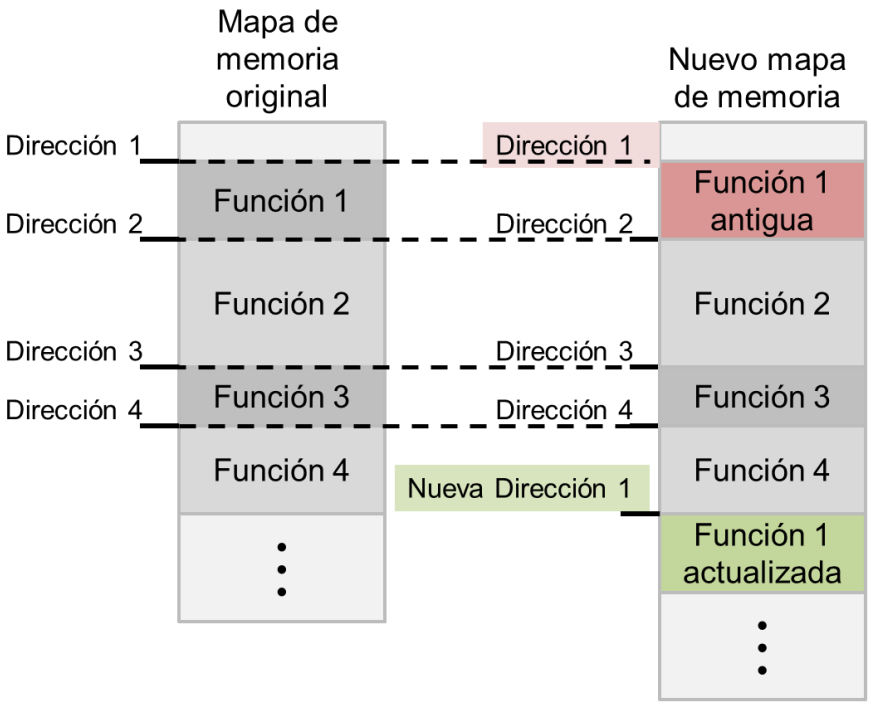


Figura 65: Propuesta de mapeo de las funciones

Un ejemplo gráfico de esta actualización parcial se puede ver en la Figura 65. Se observa a la izquierda de la figura el mapa de la memoria original y en la parte de la derecha la memoria después de la actualización. En este ejemplo se ha actualizado la Función 1. La actualización de la Función 1 se ha grabado al principio del espacio libre de la memoria *flash*, que podría ser el final de la anterior versión del firmware. Gracias a esto, se ha conseguido evitar que las funciones no actualizadas se desplacen y, por lo tanto, sus direcciones varíen. Sin embargo, habrá un problema cuando las funciones no modificadas llamen a esta nueva función en vez de a la versión antigua. Este problema se muestra en la Figura 66. En la parte de la izquierda se puede apreciar el mapa de memoria tal y como quedaría después de añadir la nueva función. Aunque tenemos en memoria almacenada la nueva

versión, ésta no está siendo utilizada por las funciones no actualizadas, ya que dichas funciones siguen llamando la dirección antigua de la Función 1 cuando quieren ejecutarla. Esto lo podemos ver en la Función 2 en la parte de la izquierda de la imagen. Para solucionar este problema se deberían modificar todas las llamadas a la Función 1 en el firmware original. Esta aproximación se presenta en la Función 2 en la parte derecha de la Figura 66. Modificar todas las llamadas en el código no modificado tiene dos inconvenientes graves. El primer inconveniente es que la solución hace crecer en tamaño el *script* de actualización ya que, además de la función nueva, habrá que actualizar las funciones que llamen a esa función. El segundo problema es debido al tipo de memoria en la que se almacena esta información. Como se explicaba en la introducción, en las memorias *flash* no se puede reescribir un byte sin borrar todo el bloque en el que se encuentra. Esto crea ineficiencia en el sistema, al tener que borrar todo el bloque de la memoria y reescribirlo casi completamente con la misma información.

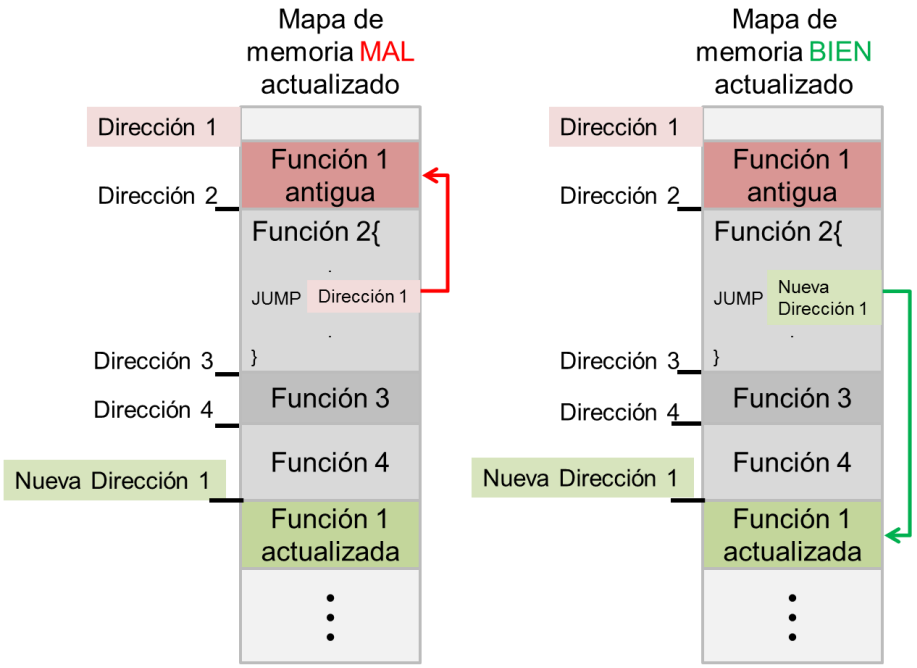


Figura 66: Propuesta de mapeo de las funciones 2

Para solucionar estas limitaciones se propone usar una tabla de referencias a funciones, la cual nos permitirá evitar el problema del desplazamiento y de reescritura completa de la memoria. Gracias a la tabla de referencias, las llamadas

a las funciones actualizadas (por ejemplo a Función 1) se podrán realizar sin cambiar las direcciones de las funciones, ya que se buscará en dicha tabla la dirección a la que se tiene que saltar (Figura 67). Por lo tanto, todas las funciones incluyen referencias a posiciones de una tabla que son independiente de la posición real de las funciones en memoria, no siendo necesario modificar la función antigua cuando se realiza una actualización. Un ejemplo del algoritmo propuesto se muestra en la Figura 67, donde se puede apreciar como la Función 1, antes de llamar a la Función 3, accede a la tabla intermedia de direcciones para consultar la dirección de la otra función y, de esta forma, poder saltar correctamente. En esta tabla esta almacenada la posición de memoria de cada función del firmware.

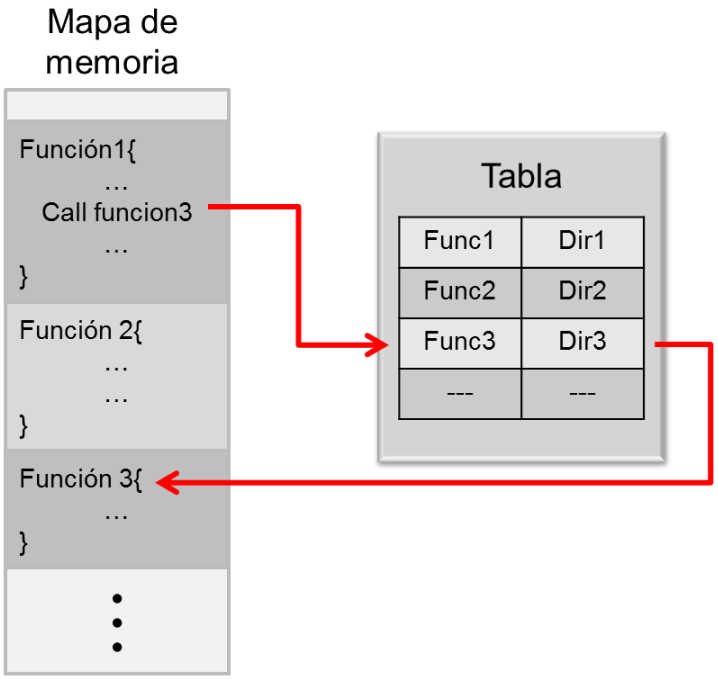


Figura 67: Tabla de direcciones en la memoria flash

Por lo tanto, si cualquier función quiere llamar a otra (en el caso de la Figura 67 la Función 1 llama la Función 3), en lugar de acceder directamente a la posición real de la función, accede a la posición de la tabla a la que corresponde, lee la dirección en la que se encuentra en ese momento y realiza una llamada a la función usando la dirección correcta. Estos saltos se hacen usando un *wrapper* genérico, que encapsula las llamadas a los objetos y permite leer la dirección final del objeto y saltar a él a través de su dirección.

En conclusión, mediante la utilización de esta técnica de actualización, no será necesario cambiar las direcciones de las funciones que llaman a las funciones actualizadas, sino que solo será necesario modificar las entradas de la tabla de direcciones intermedias para que apunte a la nueva dirección.

5.1.2.1. Actualización de la tabla de referencias

Cuando una función es actualizada con la técnica propuesta anteriormente su posición en memoria cambia. Por lo tanto, es necesario modificar las entradas de la tabla de referencias para que apunten a la nueva dirección. Esta aproximación tiene el inconveniente de obligar a reescribir en la tabla de referencias la nueva dirección de la función en la posición de memoria en la cual se almacenaba la antigua. Esto es un problema ya que la tabla está escrita en memoria *flash*, por lo que será necesario borrar completamente una sección de la memoria y volverla a reescribir modificando tan solo una dirección. Para evitar esto, se propone utilizar una solución similar a la utilizada en el caso de actualización de funciones: no modificar la misma entrada de la tabla, sino únicamente invalidarla y escribir el nuevo valor al final de la tabla.

Esta aproximación requiere usar una segunda tabla igual a la presentada en la Figura 67, que permitirá acceder a las direcciones más rápidamente. La segunda tabla estará localizada en una memoria más rápida que la memoria *flash*, por ejemplo la memoria RAM del nodo. Esta memoria puede ser leída y escrita rápidamente, sin incrementar mucho el consumo ni alterar su tiempo de vida. La estrategia propuesta copia en memoria RAM la tabla de la Figura 67 cuando el nuevo firmware se cargue o el nodo se actualice. Durante el proceso de copiado, las entradas viejas de la tabla serán eliminadas y únicamente las entradas válidas pasarán a la memoria RAM. En la Figura 68 se puede ver un ejemplo, donde la dirección de la Función 2 en la tabla almacenada en RAM es la Dir 4, en vez de la dirección original, Dir 2. Este cambio de dirección ha sido debido a que el firmware se ha actualizado (hay una nueva versión de la Función 2) y, por eso, la nueva entrada está presente, pero sin borrar la entrada de la versión anterior. Gracias a esto se evita borrar la tabla en *flash*, ya que solo es necesario marcar el valor

antiguo y añadir una nueva entrada al final de la tabla almacenada cuando sea necesario actualizarla.

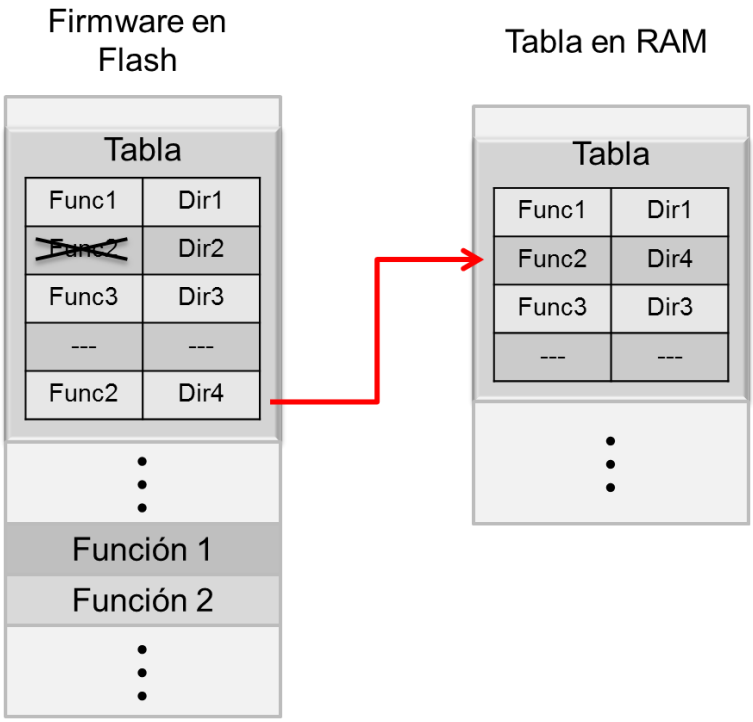


Figura 68: Colocación de la tabla de direcciones en la memoria RAM

La tabla de direcciones mostrada hasta ahora únicamente tenía dos campos: función y dirección de memoria. Sin embargo, esta aproximación no es eficiente, por lo que en la implementación del algoritmo se ha optado por implementar la tabla en *flash* con tres campos para cada objeto. En este contexto, un objeto es una función, una variable o un elemento del programa que tenga que ser relocalizado durante una actualización. El primer campo de la tabla implementada indicará la posición del objeto en la memoria (dirección). El segundo definirá si la entrada de la tabla es válida o si ha sido actualizada. El tercero señalará el índice al que se refiere la entrada. Este último campo es necesario para no perder la referencia durante la actualización.

En la parte izquierda de la Figura 69 se muestra la tabla almacenada en memoria *flash* antes de ser actualizada. Como se puede observar, el segundo campo de la tabla toma el valor FFF..FF, lo que indica que la entrada es válida.

Dicho valor también señala que esa posición de memoria no ha sido escrita después del último borrado, siendo posible escribir en ella cualquier valor diferente sin tener que borrar el sector de *flash*. Por esta razón, tras la actualización, este campo puede señalar la posición en la tabla de la nueva entrada (campo rojo de la tabla derecha de la Figura 69).

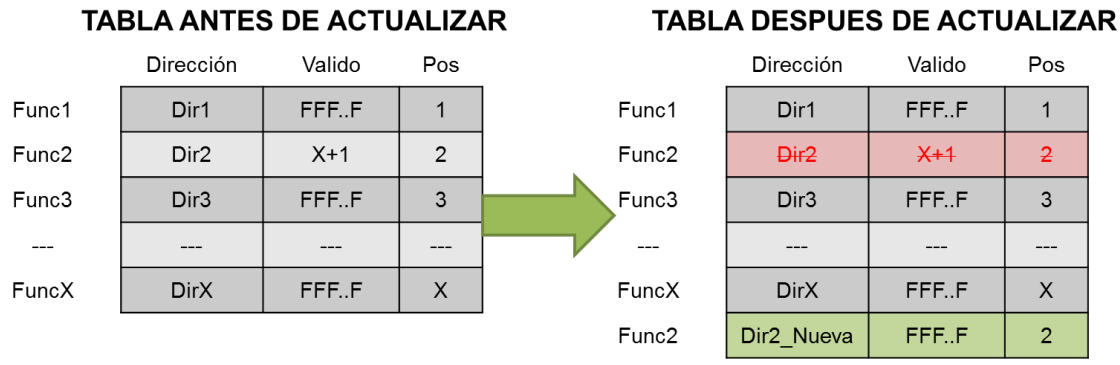


Figura 69: Tabla de direcciones completa

En el caso de que el nodo se actualice, se añadirán las nuevas entradas a la tabla y se invalidarán las anteriores entradas modificando el segundo campo. En la Figura 69 se puede observar la tabla antes y después de una actualización. Debido a que la función se ha actualizado, su dirección cambia y su referencia inicial se considerará no válida. En la tabla actualizada se puede observar que se ha creado una nueva entrada para la función Func2, la cual se ha actualizado en la posición X+1, con la nueva dirección.

En la Figura 70 se muestra un ejemplo de una tabla actualizada en *flash* y cómo es la tabla de referencias que se almacena en memoria RAM a partir de la misma. Se puede apreciar que el tamaño de la tabla almacenada en RAM es inferior a la que está en *flash*, debido a que la primera únicamente contiene la dirección final de cada función y se han eliminado las entradas no usadas. En la tabla localizada en la memoria RAM, cada función se asocia a un índice de la tabla.

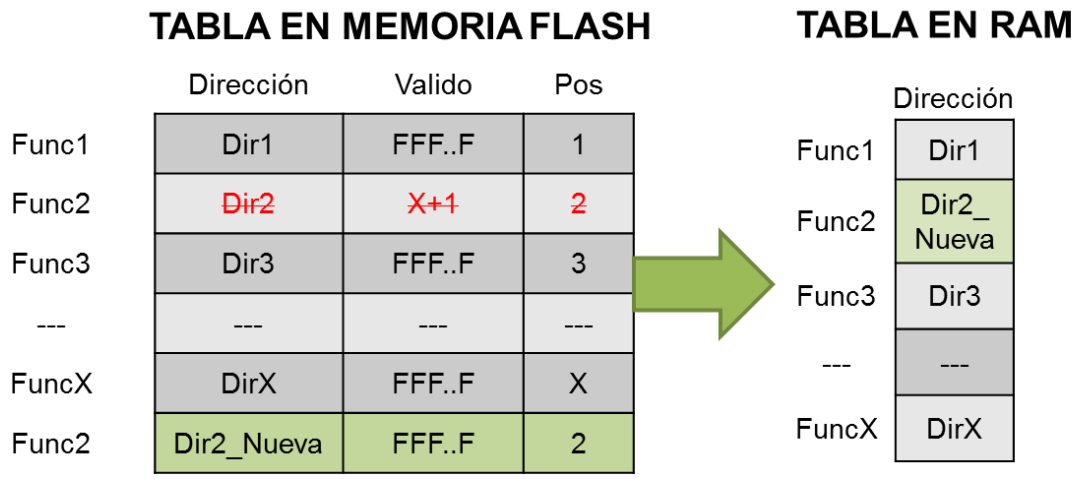


Figura 70: Tabla de direcciones almacenada en memoria RAM

Una extensión de esta técnica permite añadir control de versiones, como se verá en la sección siguiente. Esta extensión permite ejecutar versiones antiguas de firmware en caso de que la actualización falle.

5.1.3. Extensión de la técnica para control de versiones

Gracias a la técnica de actualización de los datos propuesta, la información de todas las versiones del firmware está guardada en la memoria del nodo. Por ello, es posible tener un control de versiones que permita ejecutar fácilmente versiones anteriores del firmware, sin transmitir información adicional por la red. Para ello, solo es necesario hacer una pequeña modificación en la tabla de referencias, añadiendo la versión de cada función. Como se puede observar en la Figura 71, la tabla almacenada en *flash* almacena todas las direcciones de las versiones previas que el nodo tiene disponibles. Cada entrada de la tabla en *flash* tiene un campo que almacena la versión en la que fue actualizada esa función (tercer campo). Si se necesitase ejecutar la última versión almacenada en el nodo, la tabla que se copie a RAM contendrá las direcciones de las últimas versiones de cada función. Sin embargo, si se quiere utilizar una versión previa, se pueden copiar únicamente las

direcciones con el valor de campo menor o igual que la versión deseada. La Figura 71 muestra dos ejemplos de cómo quedarían las tablas en la memoria RAM después de seleccionar dos versiones distintas. En el primer caso, la RAM tiene las entradas de las última versión actualizada (La dirección de la Func2 es DiR2_Vx). En el segundo caso, la versión seleccionada es la Version2, y la dirección de la función es Dir2-V2.

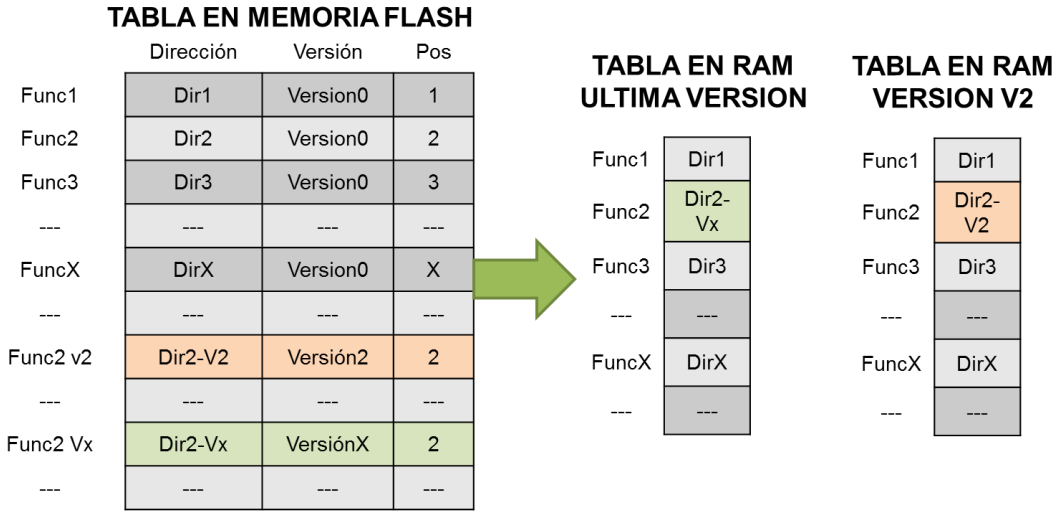


Figura 71: Tabla de direcciones con control de versiones

5.1.4. Proceso de actualización de los nodos

Hasta ahora se ha presentado la nueva técnica de actualización parcial, comentando detalles de su funcionamiento en el nodo. Sin embargo, la técnica de actualización parcial o incremental precisa de una serie de herramientas externas al nodo que ayuden a generar toda la información que se necesita para la modificación del software. En la Figura 72 se presenta un diagrama completo del esquema propuesto para generar la actualización de los nodos de la red, empezando con la comparación de las dos versiones de firmware (la versión de partida o antiguo firmware y la nueva) y concluyendo con la ejecución de la actualización.

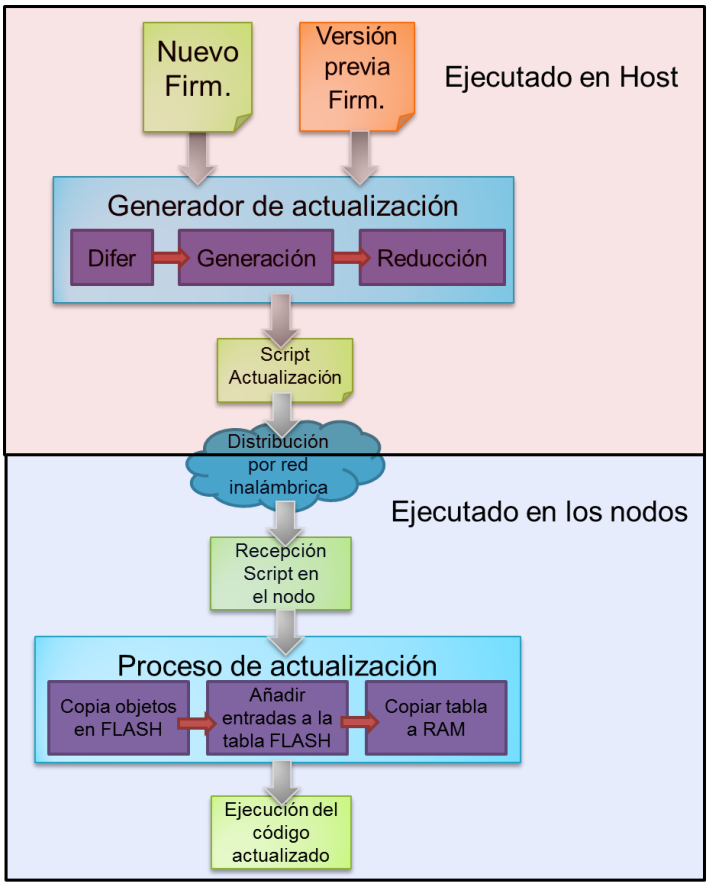


Figura 72: Diagrama del proceso de actualización a alto nivel

El primer paso de este proceso consiste en la generación del *script* de actualización. Para ello, se comparan el firmware viejo con el nuevo. Con las diferencias entre ambos firmwares, se genera el *script* de actualización y se intenta que tenga el menor tamaño posible. Normalmente, el tamaño del *script* es mucho menor que el tamaño del firmware completo. Es entonces cuando, a través de un Gateway o un nodo de la red, se distribuye el firmware con un protocolo de distribución de actualizaciones a los nodos que tienen que ser actualizados. Una vez que los nodos reciben el *script* completo, pueden empezar el proceso de actualización. Es en esta etapa cuando se añaden las nuevas funcionalidades al final de la memoria *flash* y se insertan las nuevas entradas en la tabla de referencias. En este momento estaría la actualización cargada, pero aún no estaría funcionando, ya que la tabla de direcciones en RAM (que es la que se utiliza durante la ejecución del SW) tiene almacenadas las direcciones de la versión

anterior. Cuando se actualice la memoria RAM (normalmente reiniciando la aplicación), la nueva versión estará completamente operativa.

5.1.4.1. Generación automática de la actualización

El proceso de preparación del código de la aplicación para que pueda ser actualizado mediante la técnica propuesta es bastante complejo.

El primer paso es la ordenación de las funciones en la memoria *flash*. Esta ordenación es un proceso tedioso, debido al gran número de funciones que un firmware puede llegar a contener. Además, el desarrollador necesita conocer el tamaño de cada función para poder organizar la memoria. Con las funciones correctamente ordenadas es preciso generar tres ficheros diferentes. El primer fichero contiene las direcciones de cada función para que esta información pueda ser añadida en el *script* de *linkado*. El segundo fichero (que hemos llamado "functions_table.c") contiene la tabla con las funciones y las funciones *wrapper* que permiten saltar a las direcciones de la tabla. El tercer fichero ("functions_table.c") contiene funciones de soporte necesarias.

El segundo paso en el proceso de generación de la actualización consiste en añadir estos tres ficheros al proyecto con la aplicación, de forma que se integren en el firmware del nodo. Una vez que la aplicación está compilada, el firmware puede ser cargado y ejecutado en el nodo.

El último paso sería la generación de una actualización parcial. Para ello, se compila el nuevo firmware y se genera una imagen ejecutable (un fichero con formato hexadecimal, .hex). A continuación, se realiza una comparación entre la versión antigua y nueva de firmware. Para obtener una actualización incremental eficiente, el firmware nuevo debe ser generado con las menores diferencias posibles con la versión anterior. Por esta razón, se ordenan las funciones en memoria de la misma forma en diferentes versiones de firmware, para que coincida su secuencia en *flash*. Gracias a esta estrategia es posible reducir el tamaño del *script* de actualización. En este trabajo se realiza de forma automática la

comparación de firmwares y la generación de la secuencia de comandos de actualización (*script* de actualización). Este *script* es la clave para reconstruir el nuevo firmware en el nuevo nodo.

5.1.4.1.1. Integración del proceso de actualización en el entorno de desarrollo

Debido a la complejidad del proceso de generación de la actualización, ha sido necesario diseñar una herramienta que facilite el uso de este proceso hasta el punto de hacerlo casi invisible al desarrollador final. Para ello, se ha desarrollado un *plug-in* para Eclipse [127] que facilita la inclusión de la técnica de actualización parcial en el entorno de desarrollo software (SDK, *Software Development Kit*) de la aplicación del nodo. Este *plug-in* permite generar automáticamente las actualizaciones desde la misma herramienta que se está utilizando para desarrollar el software del nodo. Tras la instalación del *plug-in* desarrollado en el SDK aparece un nuevo menú en el IDE de desarrollo software. Este menú se llama "*Partial Update*", y aparece en la Figura 73. Como se puede ver, hay 5 opciones diferentes en el menú. La primera opción (*Configure Project*) abre una ventana que permite configurar diferentes opciones del proyecto. La segunda opción (*Show Memory*) es una ventana con áreas seleccionables (*checkbox)* que permite al usuario ver como se ordenan las funciones en la memoria al tiempo que permite hace una ordenación manual. La tercera opción (Compile) compila el proyecto introduciendo automáticamente las fuentes adicionales necesarias para permitir la actualización parcial al proyecto. Esta opción genera la imagen del firmware que sería cargada en el nodo. La cuarta opción (*Generate Update*) compila el proyecto y genera un fichero de pequeño tamaño con la actualización. Finalmente, la última opción (*Clean*), limpia el proyecto eliminando los ficheros temporales.

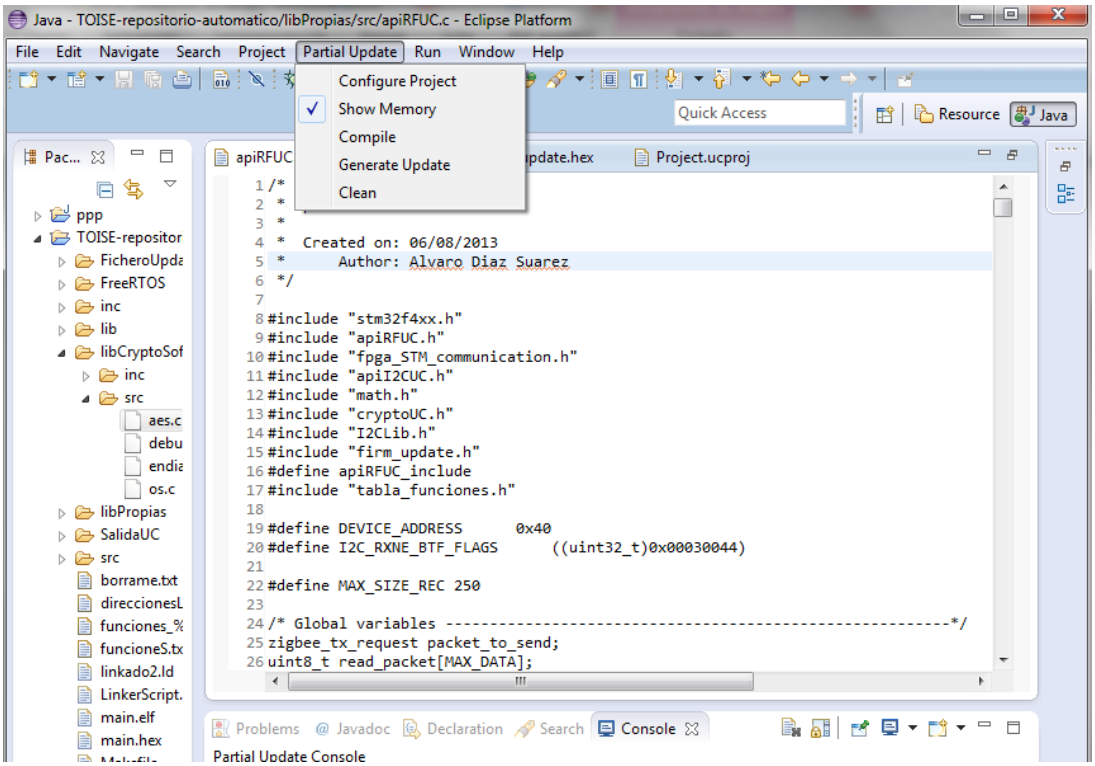


Figura 73: Plug-in para facilitar la actualización parcial

La opción "*Configure Project*" lanza una ventana de configuración (Figura 74). En esta ventana es importante definir algunos atributos del proyecto para permitir generar correctamente los ficheros de *linkado* y orden de compilación (*Make)*. También es necesario definir los directorios donde se encuentran los ficheros que deben de ser incluidos del proyecto y las opciones de compilación. Además tenemos la posibilidad de definir distintas utilidades como el Compilador o el *Linker* que queremos utilizar. También será necesario definir el tamaño y las direcciones de la memoria *flash* donde queremos alojar los objetos. Entre ellos, es importante indicar la dirección donde vamos a guardar la tabla, el código y el *wrapper*. También tendremos que definir las características de la RAM. Esta configuración puede ser guardada como características del proyecto con el botón "*Save configuration*", el cual genera un fichero de proyecto denominado "Project.ucproj".

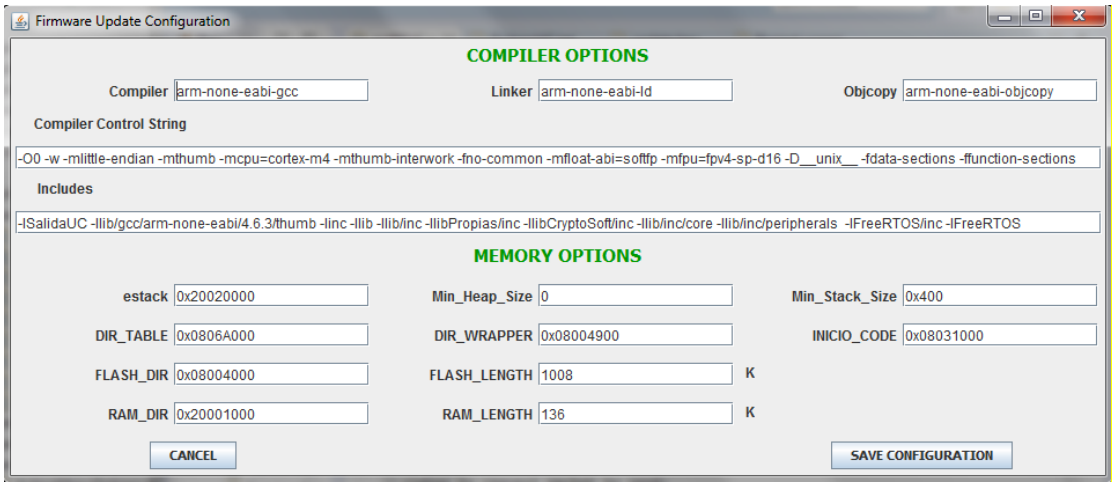


Figura 74: Ventana de configuración del proyecto

Una vez que el proyecto está configurado, se inicia el proceso de compilación y generación de la actualización. La Figura 75 representa un diagrama con los pasos realizados por la herramienta durante el proceso de compilación. Como se puede observar, los primeros pasos consisten en la generación de los ficheros de *linkado* y compilación. Estos ficheros permiten compilar el proyecto y obtener la información de la imagen del firmware, como son los tamaños de cada elemento de memoria. Gracias a esta información, el *plug-in* podrá generar una ordenación del mapa de la memoria eficiente. Cuando la memoria esté correctamente ordenada, se generarán automáticamente los ficheros adicionales que permitirán usar la tabla de direcciones comentada en las secciones previas. Estos ficheros contendrán las direcciones de cada elemento de la memoria. Además de la generación de estos ficheros, los ficheros del proyecto se analizarán (*parseado*) y se incluirán nuevas instrucciones para soportar la actualización parcial. A continuación, se generarán un nuevo *Makefile* y *script* de *linkado,* que serán los encargados de posicionar todos los elementos en las direcciones de memoria seleccionadas durante el proceso de compilación. Una vez que todos estos elementos se han generado correctamente, el proyecto se recompilará y se generarán principalmente dos ficheros. El primer fichero es la imagen del firmware con la tecnología que permite la actualización parcial incluida (fichero hex) y el segundo será un fichero de configuración (ConfigFile.uc) que será necesario para generar fácilmente actualizaciones del sistema.

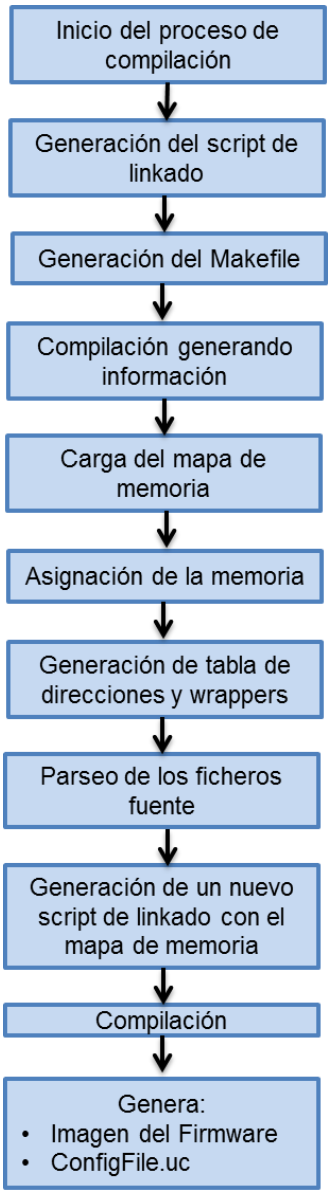


Figura 75: Diagrama del proceso de compilación

La otra tarea realizada por el *plug-in* es la generación de las actualizaciones. Este proceso es más complicado que el proceso de compilación (Figura 75). El primer paso en el proceso de generación de una actualización es la carga del fichero de configuración con la información del firmware cargado en el sistema (Fichero ConfigFile.uc generado por el proceso de compilación o por la última actualización llevada a cabo en el nodo). El siguiente paso del proceso es la generación de un *Makefile* y de un *script* de *linkado* que permiten extraer información de los elementos actuales del firmware a compilar. Con esta nueva

información se genera un nuevo mapa de memoria para el nuevo firmware. Una vez que tenemos ambos mapas de memoria generados (firmware antiguo y nuevo), estos son comparados buscando cambios en la memoria. Esta primera comparación, debido a que no se pueden saber los posibles cambios que ha habido en la funcionalidad, se realiza comparando los tamaños de cada elemento. Cada cambio encontrado dentro de un elemento deberá ser realojado en una nueva posición de memoria libre. Es entonces cuando se generan y se analizan (*parsean*) los ficheros del proyecto de manera similar a la explicada en el proceso de compilación. Cuando todos estos ficheros están generados, el compilador genera una nueva imagen de firmware completa que será comparada con la imagen alojada en el momento de la actualización en el sistema (y en el fichero de configuración configFile.uc). La idea de esta comparación es detectar cambios que no hayan podido ser detectados en la comparación previa. En estas comprobaciones no se mira el tamaño de los elementos, sino cada byte de información de los elementos. Esto es útil porque puede darse el caso de que en una actualización se haya cambiado únicamente el valor de ciertos datos y esto no afecte al tamaño final del elemento (por ejemplo, cuando en una función se cambia el valor de una variable). Si se detecta que hay cambios adicionales, el flujo de la Figura 76 salta al paso anterior "*allocate memory*" y vuelve a generar todos los archivos para volver a comparar. Cuando todos los cambios han sido detectados y realojados en la memoria, el proceso finaliza generando el fichero reducido de actualización y el nuevo fichero de configuración, que será necesario para posteriores actualizaciones.

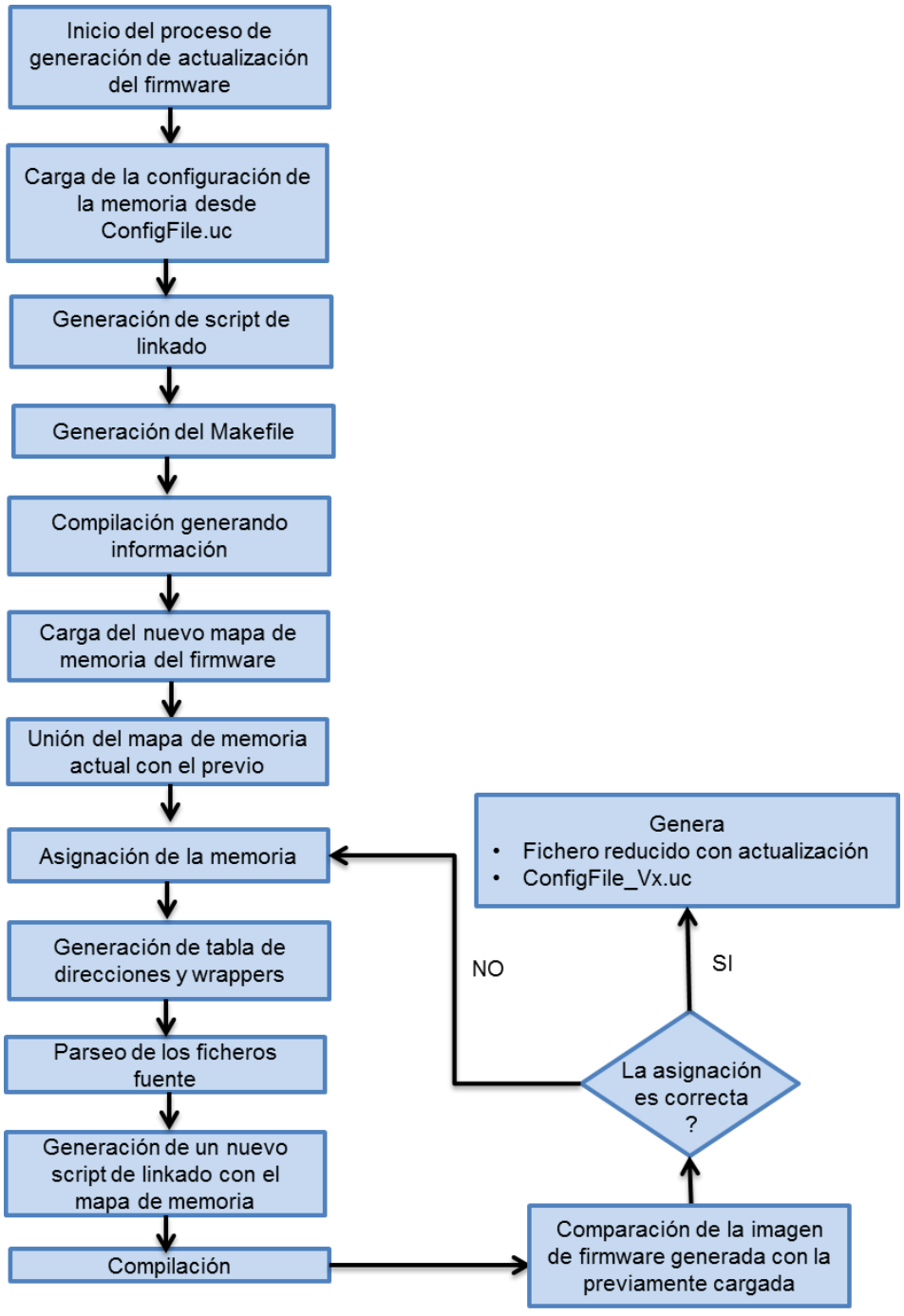


Figura 76: Diagrama del proceso de generación de la actualización parcial

En el caso de que se seleccionase la opción "*Show Memory*" durante el proceso de generación, se mostrará una nueva ventana con el mapa de memoria propuesto (Figura 77). En esta ventana será posible recolocar y reconfigurar el mapa de memoria propuesto inicialmente por el *plug-in*. Los usuarios podrán buscar los

objetos a través de su dirección o su nombre y decidir una nueva ubicación para ese objeto. Además, está disponible el botón "*Compact*" que nos permite eliminar los espacios no utilizados de la memoria para un mejor aprovechamiento.

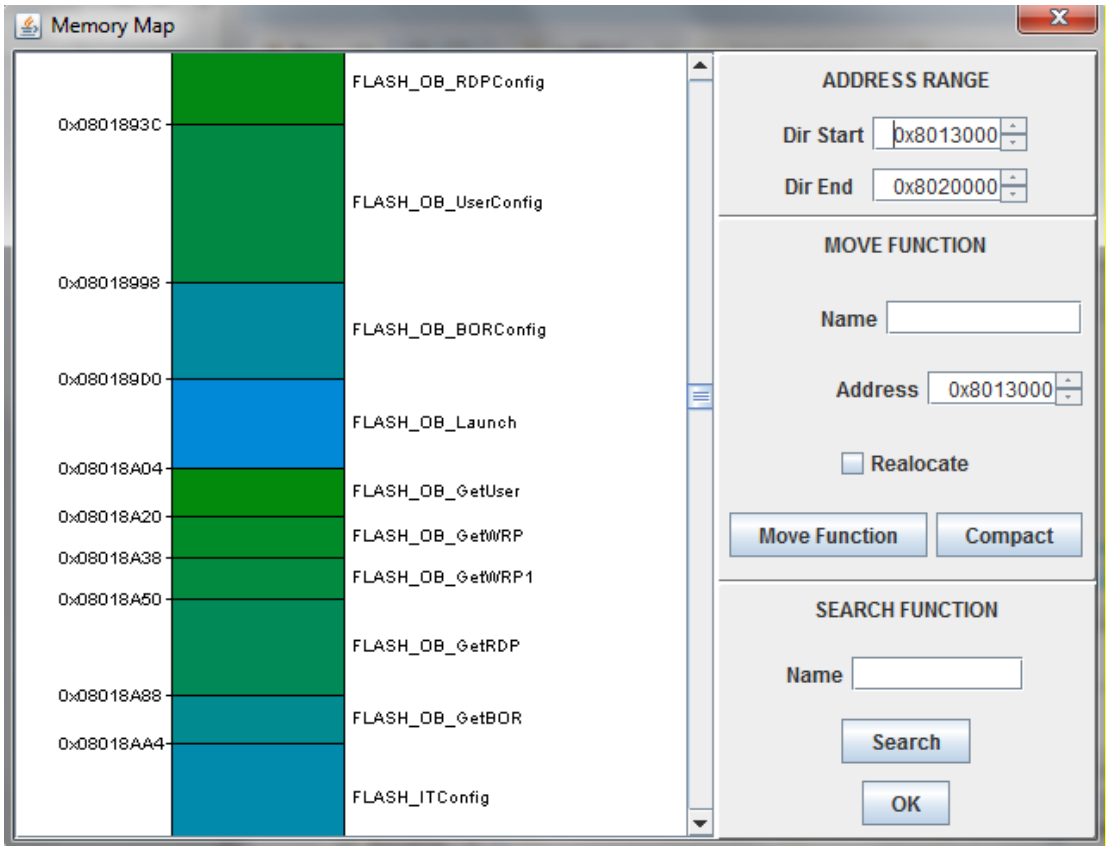


Figura 77: Ventana para ver/modificar la ordenación de la memoria

5.1.5. Validación de la metodología

Para poder validar y demostrar la eficacia de la tecnología propuesta, se han realizado distintas pruebas de campo. Para ello, se ha montado una red de sensores inalámbrica como se puede ver en la Figura 78. Dicha red está compuesta por cuatro nodos. Los tres nodos finales son los que se actualizan y el cuarto nodo es el Gateway. El Gateway es el encargado de recibir la actualización a través de un módulo GPRS y transmitirla a los nodos finales usando un protocolo de transmisión de actualizaciones (OTAP) que se ha implementado para el

demostrador, pero que está fuera del alcance de este trabajo. La arquitectura de los nodos está compuesta principalmente por distintos sensores, un LCD y una plataforma Silica Xynergy-M4 [128]. La red inalámbrica es de tipo Zigbee.

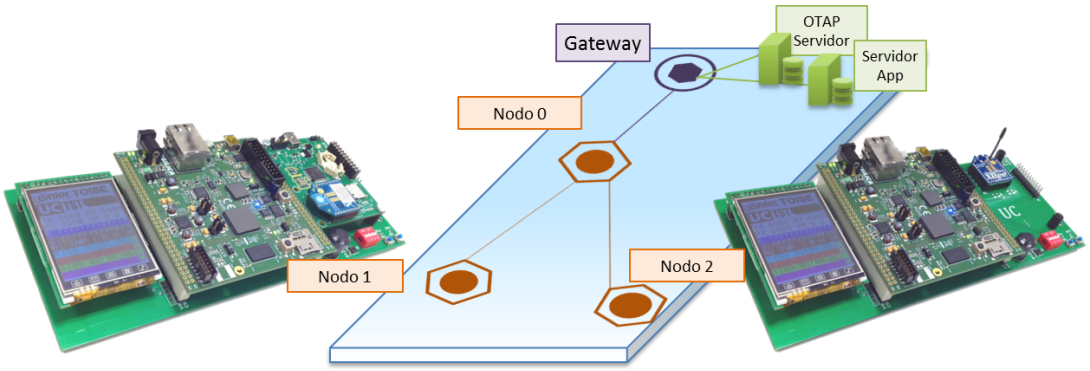


Figura 78: Red desplegada para probar la actualización parcial

Debido al protocolo Zigbee y al modelo de paquete diseñado para usar en esta red durante la actualización del firmware (limitado por el protocolo de transmisión de la actualización), únicamente se pueden transmitir paquetes con 76 bytes de datos. Por lo tanto, si un firmware tiene un tamaño medio de 130 Kb (el cual podría ser un tamaño típico para un firmware pequeño) esto significa que es necesario transmitir más de 1700 paquetes de datos para cada nodo (sin tener en cuenta las retransmisiones debido a fallos en la recepción de paquetes). Asumiendo que una tasa de transmisión normal puede ser de un paquete por segundo (según las mediciones realizadas, es posible incrementar esta tasa pero con el coste de incrementar los problema en las transmisiones), esto significa que una actualización tardaría más de 30 minutos en el mejor escenario (en laboratorio, con alta potencia de transmisión y sin interferencias). Esto implica además de una gran latencia, un alto consumo de energía en el sistema. Como se ha comentado, la actualización sin usar la técnica propuesta implicará, en el mejor de los casos, una detención de la funcionalidad de la red durante el tiempo de transmisión del nuevo firmware. La alternativa más comúnmente utilizada para evitar estas paradas es la transmisión de la actualización conjuntamente con los datos de la red, lo que implica que el tiempo total de actualización incrementa a varias horas o incluso días.

Es complicado evaluar esta técnica, debido a que el tamaño de la actualización es proporcional al número de cambios que tiene el firmware actualizado respecto al residente en el sistema y al tamaño de estos. Pero, por ejemplo, para un cambio en una única función que corrige un bug, la actualización con la técnica propuesta podría tener un tamaño menor a 600 bytes. Esto nos permitiría tener que transmitir únicamente 9 paquetes. Si lo comparamos con los 1700 paquetes que sería necesario transmitir en caso de realizar el mismo cambio, pero sin la técnica propuesta, se observa que se obtiene una diferencia del tiempo de transmisión muy grande (unos 10 segundos frente a los 30 minutos).

Se han realizado distintas pruebas para confirmar estos datos. La evaluación ha consistido en transmitir el nuevo firmware a los dispositivos reales usando la red presentada en la Figura 78. El nuevo firmware transmitido corrige algunos bugs introducidos en el firmware grabado en los nodos. Se han usado dos casos distintos. En el primer caso, el firmware únicamente corrige un bug. Sin embargo, en el segundo caso, se corrigen 10 errores. Por lo tanto, el tamaño total de las actualizaciones parciales ha variado, mientras que en el caso de las actualizaciones totales el tamaño de las transmisiones es siempre el mismo. Durante la transmisión de las actualizaciones, se han medido distintos aspectos como, por ejemplo, el tamaño del firmware transmitido, cuántos paquetes han sido necesario enviar, el número de retransmisiones que han ocurrido mientras se transmitían las actualizaciones o el nuevo firmware y el tiempo total necesario para que el firmware actualizado esté cargado y disponible.

En la Tabla 24 se pueden observar los resultados obtenidos en las pruebas realizadas. Usando la técnica de actualización parcial propuesta, el tiempo del proceso, el tamaño de la actualización, el número de paquetes y el número de retransmisiones es siempre mucho menor comparándolo con una actualización realizada sin usar el procedimiento propuesto.

Tabla 24: Pruebas de actualización del firmware

	¿Actualización realizada correctamente?	Tamaño de la actualización	Numero de paquetes	Numero de Retransmisiones	Tiempo total
Firmware Total (10 bugs)	**SI**	130 Kbytes	1752	102	31 min
Firmware Total (1 bug)	**SI**	130 Kbytes	1752	105	31 min
Firmware Parcial (10 bugs)	**SI**	4.8 Kbytes	65	4	1 min 11 s
Firmware Parcial (1 bug)	**SI**	593 Bytes	8	0	10 s

5.2. Arranque seguro

Además de la técnica de actualización parcial presentada en la sección anterior, en esta tesis se ha desarrollado un segundo servicio de la plataforma software: el arranque seguro.

Muchas redes de sensores inalámbricas procesan información sensible, por lo que es necesario asegurar que no ejecutan código malicioso. Una de las formas típicas de introducir código malicioso en un nodo es atacar el proceso de arranque del sistema (*booting*). Para contrarrestar estos ataques y reducir las vulnerabilidades del sistema, los nodos de las redes tienen que ejecutar siempre un firmware (software) legítimo y a prueba de manipulaciones. Para ello, el sistema tiene que estar seguro de que el firmware que ejecuta es válido, legítimo o firmado por alguien de confianza y que no ha sido alterado/cambiado por atacantes.

Existen muchos aspectos que tienen que ser comprobados para garantizar que el firmware es seguro: desde que el código que se va a ejecutar no esté corrupto o intencionadamente modificado por un atacante hasta que el software esté alojado

en una zona segura a la que un atacante no pueda acceder. En esta sección nos centraremos en asegurar la integridad y autenticidad del firmware que queremos ejecutar, sin incrementar significativamente el consumo de potencia del nodo de la red inalámbrica.

Para garantizar la seguridad del firmware normalmente es necesario encriptar o desencriptar código, con el consiguiente incremento del consumo de energía. Para evaluar el impacto de este proceso, se va a medir el consumo de diferentes algoritmos criptográficos, comparando las prestaciones de la implementación software frente a las de módulos hardware específicos. Con los resultados obtenidos se seleccionarán los algoritmos con mejor relación seguridad/consumo para implementar el proceso de arranque.

El proceso de arranque propuesto es una mejora de la técnica normalmente utilizada, y que es mostrada en el diagrama de la Figura 79. Este proceso es el responsable de acceder a la memoria en donde el firmware está almacenado, extraerlo y ejecutarlo. Antes de descargar el firmware de la memoria, nos debemos asegurar que la fuente de la que proviene el software es de confianza y que, por ello, el código es legítimo. También tenemos que asegurar la integridad del firmware, es decir, debemos comprobar que el código no se ha corrompido por problemas de almacenamiento y/o acceso a memoria. Una opción típicamente utilizada para asegurar la integridad del firmware en el arranque es añadir una firma al código que queremos cargar. Dicha firma es verificada antes de cargar el firmware. Después de esta comprobación, en el caso de que todo haya sido validado correctamente, la carga del firmware puede ser realizada. Este procedimiento básico de arranque, que usan la mayoría de redes de sensores inalámbricas, tiene un problema importante: aunque se asegura (de una manera débil) la integridad del firmware, no se garantiza la confidencialidad. Por lo tanto, un atacante podría insertar un código malicioso y recalcular la firma con relativamente poco esfuerzo. Este tipo de ataques se podría evitar encriptando el código que queremos cargar. Es importante recordar que, en redes de sensores inalámbricas, uno de los requisitos esenciales a la hora de diseñar el arranque seguro es que este no incremente significantemente el consumo de los nodos. Por ello, será

necesario realizar un estudio de los métodos criptográficos existentes en el estado del arte así como su implementación en redes de sensores inalámbricas.

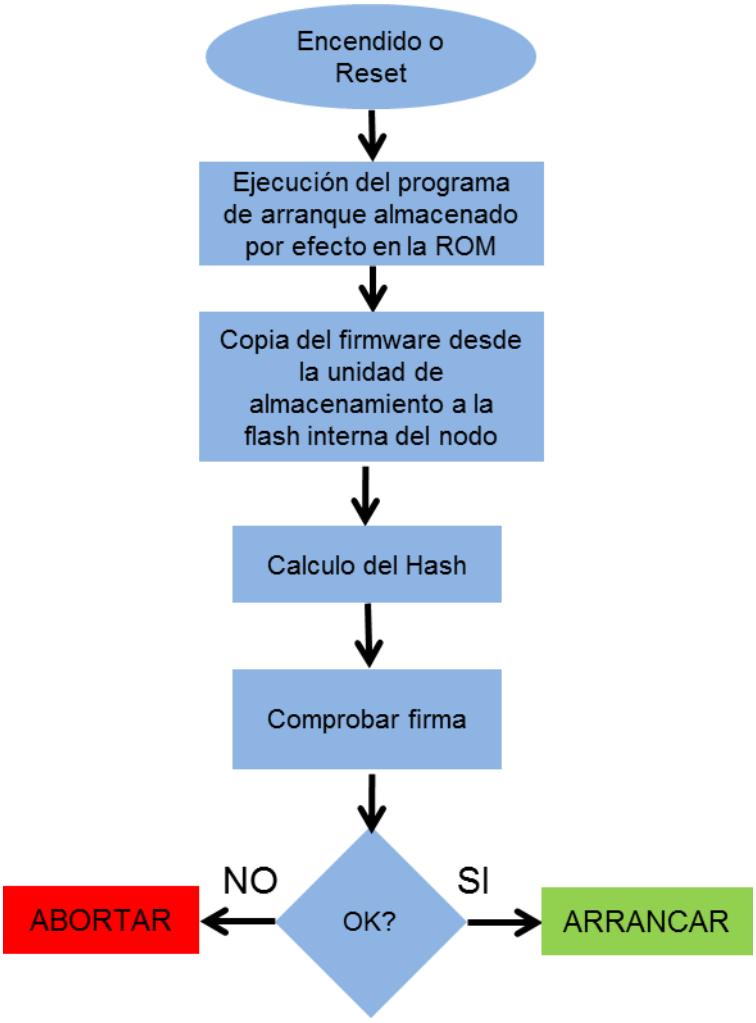


Figura 79: Arranque de sistema genérico con mínima protección

5.2.1. Arranque seguro: Estado del arte

La seguridad del proceso de arranque es un tema que ha sido ampliamente estudiado. Sin embargo, no existen tantos estudios sobre la seguridad en este proceso en redes de sensores inalámbricas y sistemas embebidos. Uno de estos trabajos es el presentado en [129], donde los autores presentan una técnica de

arranque seguro para plataformas embebidas. Aunque se hace mención a un módulo criptográfico y al manejo de las claves, no se proporciona información específica sobre el método utilizado. En [130] los autores ponen de manifiesto la importancia de proteger los nodos que componen las redes de sensores inalámbricas durante el proceso de arranque. En [131] se presenta una “cadena de confianza” para solucionar el problema del proceso del arranque seguro. En todos estos trabajos se muestra la importancia de asegurar la confidencialidad y la integridad del firmware que el sistema debe ejecutar y para ello es normalmente necesario utilizar algoritmos criptográficos. En la sección 4.1 de esta tesis se realizó un análisis de algunos aspectos de los algoritmos criptográficos. Sin embargo, en dicha sección no se tuvo en cuenta el impacto que tienen estos algoritmos en redes de sensores inalámbricas. La importancia de medir el consumo de las técnicas de encriptación en las redes de sensores inalámbricas es señalada en [132]. Sin embargo, en esa publicación solo estudian métodos basados en AES. En [133], los autores realizan un amplio análisis del coste de usar algoritmos criptográficos simétricos y asimétricos así como funciones hash, pero solo en dispositivos portátiles. En [134] se presenta un novedoso esquema de gestión de claves orientado a la seguridad de redes de sensores inalámbricas. El artículo está basado en un método de clave pública y se centra en la metodología para crear y compartir claves para encriptar (como RSA o ECC), aunque no incluye resultados prácticos que confirmen el esquema propuesto. Los métodos criptográficos pueden ser clasificados en procedimientos simétricos y asimétricos, dependiendo de si existe o no una clave pública. El arranque seguro que se propone en esta tesis utiliza encriptaciones simétricas debido a tres importantes razones:

- Los métodos simétricos normalmente tienen un consumo de energía menor que los métodos asimétricos. Estos aspectos se analizan en [133] y [105].

- Hoy en día es corriente que las plataformas incluyan periféricos hardware específicos para encriptación simétrica como AES (*Advanced Encryption Standard*) y triple DES (*Data Encryption Standard*).

- La clave privada es almacenada en el dispositivo durante la fase de inicialización de la red, lo que elimina el proceso de distribución de claves y evita incrementar el consumo de los nodos de la red.

Como ya se ha comentado anteriormente, los dos aspectos más importantes a la hora de asegurar un arranque seguro son la autenticación y la confidencialidad. Para autenticar el firmware, se puede usar un algoritmo HASH para obtener una firma MAC (*Message Authentication Code*). Dependiendo del nivel de seguridad deseado, el algoritmo HASH puede ser protegido con un *password* (HMAC, código de autenticación de mensajes basado en HASH). Esta firma normalmente se coloca al final de la imagen del firmware. Cuando el nodo descarga el firmware, calcula el HASH de la imagen y lo compara con el HASH incluido en el código. Si los dos son iguales, podremos decir que el firmware esta autentificado. Como ocurre con la encriptación, los sistemas embebidos suelen ofrecer módulos hardware específicos para hacer los cálculos del HASH. En el caso del dispositivo STM32F4, la funcionalidad del Hardware de cálculo de HASH incluye MD5 (*Message Digest Algorithm* 5) y SHA-1 (*Secure HASH Algorithm* 1). La diferencia entre ambos radica en el tamaño de la firma generada. MD5 crea un HASH de 16 bytes mientras que en el caso de SHA-1, el tamaño del HASH resultante es de 20 bytes. Por lo tanto se espera que SHA-1 requiera un poco más de consumo que MD5, el cual provee un poco menos de seguridad.

Para proveer integridad y confidencialidad simultáneamente, el firmware tiene que ser encriptado. Para ello se pueden utilizar algoritmos como AES (*Advanced Encryption Standard*), que sustituyó a DES (*Data Encryption Standard*) como técnica más utilizada. Aplicando el algoritmo DES tres veces (algoritmo 3DES o TDES, Triple DES) se obtiene un resultado más seguro. AES es un método criptográfico estándar de clave simétrica que genera eficientemente un texto cifrado partiendo de un texto plano mediante el uso de una clave privada. Su eficiente implementación es debida a que en el algoritmo se realizan únicamente operaciones a nivel de bit, como XOR o desplazamientos. El tamaño de bloque para el algoritmo AES es de 128 bits. Las claves pueden tener un tamaño de 128 (AES-128), 192 (AES-192) o 256 bits (AES-256). Cuanto mayor sea la clave, más seguridad tendrá el texto cifrado.

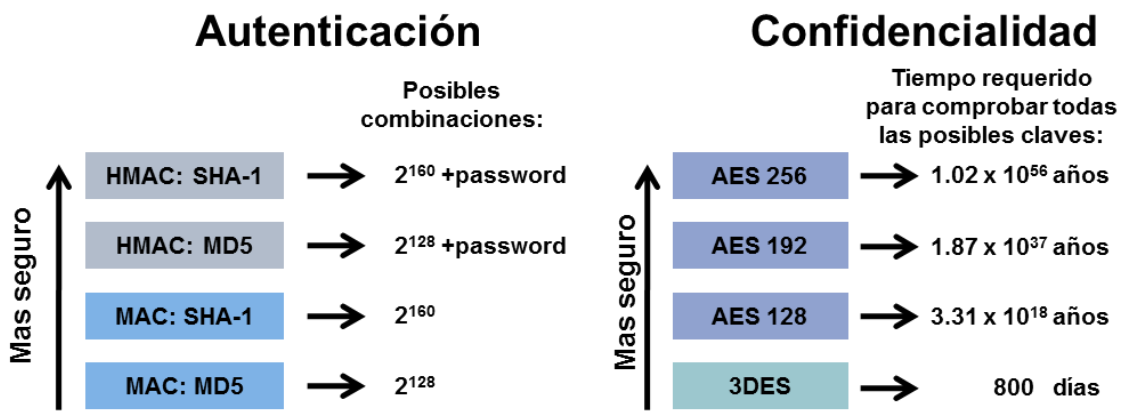


Figura 80: Seguridad de los distintos métodos

Respecto al rendimiento de estos algoritmos de autentificación y encriptación, en [105] se puede observar que AES-128 es más seguro que TDES y que el SHA-1 es más resistente frente a colisiones (probabilidad de obtener el mismo HASH para dos imágenes distintas) que el MD5. En la Figura 80 se muestra una comparación de la seguridad (obtenida de [134] y [135][86]) de los distintos métodos criptográficos.

Como ya se ha comentado, no solo es importante que el arranque seguro garantice la autentificación y confidencialidad del firmware, también es esencial que no incremente el consumo del nodo. Existen trabajos anteriores sobre estimación del rendimiento de criptografía en sistemas embebidos. En [133], el análisis concluye que un hardware específico normalmente obtiene la mejor relación entre tiempos de ejecución y consumo. Sin embargo, es deseable ejecutar algunas pruebas en el dispositivo para poder verificar estos resultados en la plataforma de destino. Esta tarea se realizará en la siguiente sección.

5.2.2. Medidas de consumo de los algoritmos criptográficos

El consumo de los algoritmos criptográficos es una característica muy importante a tener en cuenta en las redes de sensores inalámbricas. Para poder seleccionar los algoritmos con mejor relación seguridad/consumo se van a realizar medidas reales sobre un dispositivo con el equipamiento y las técnicas adecuadas. Estas medidas mostraran el consumo de los diferentes esquemas criptográficos que normalmente se utilizan en el proceso de arranque. Además, en el caso de que el firmware este firmado (autentificación de código), el proceso de arranque tendrá que calcular el HASH del firmware recibido y comparar la firma para asegurar la autenticación.

Hoy en día, varias plataformas hardware, como la familia STM32F4 de STMicroelectronics, incluyen módulos criptográficos. Para realizar las medidas y probar el método de arranque propuesto se ha utilizado la plataforma Silica Xynergy-M4 [128], que integra un dispositivo STM32F4 con un procesador ARM Cortex-M4. Además, el dispositivo incluye módulos de criptografía simétrica, como el TDES (Triple DES), el AES-128 (AES con clave de 128 bits), AES-192 y el AES-256. Estos algoritmos han sido implementados tanto en hardware como en software, para comparar tiempos de ejecución y consumos. Las medidas que se han tomado son específicas para la plataforma Silica Xynergy-M4 [128], pero el rendimiento de los algoritmos en otras plataformas embebidas se asume que es similar.

La Figura 81 muestra los tiempos de ejecución y consumo obtenidos para los diferentes algoritmos en la plataforma anteriormente reseñada. Los resultados se han obtenido con un tamaño de firmware de 25600-bytes y a una frecuencia de reloj de 120 MHz. Los resultados de encriptación y desencriptación son similares, por los que solo se presentan los de encriptación.

Como se puede observar, la implementación del AES en hardware es unas 10 veces más rápida que la implementación en software. La diferencia en el consumo

mantiene la misma relación. En el caso del TDES, la implementación en software es mucho más lenta y consume mucha más energía que la implementación hardware (aproximadamente un orden de magnitud). Además, si comparamos el consumo de la implementación software del TDES con la de los AES, se observa que la misma es 7 veces mayor para el caso de AES-128 y 5.4 veces mayor que el AES-256. En el caso de la implementación en hardware, el consumo del TDES es 3 veces mayor que el AES-128 y 2.2 veces mayor que el AES-256.

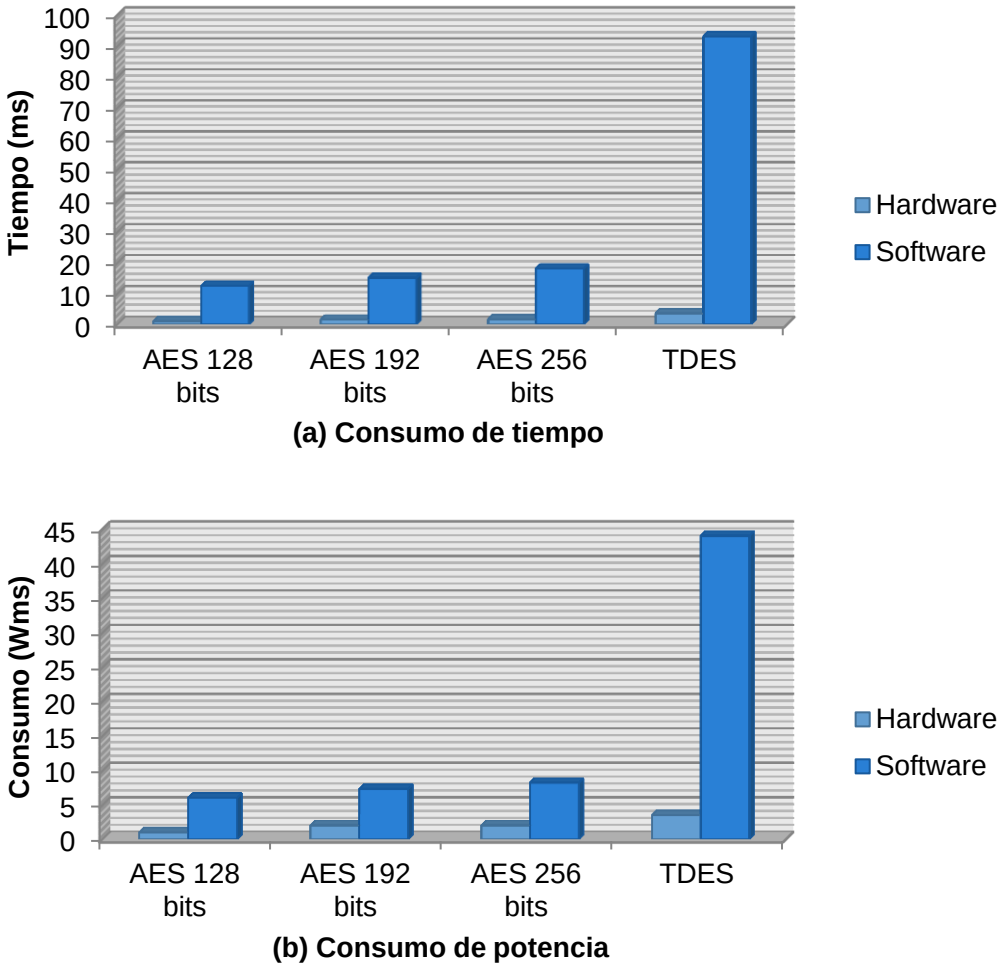
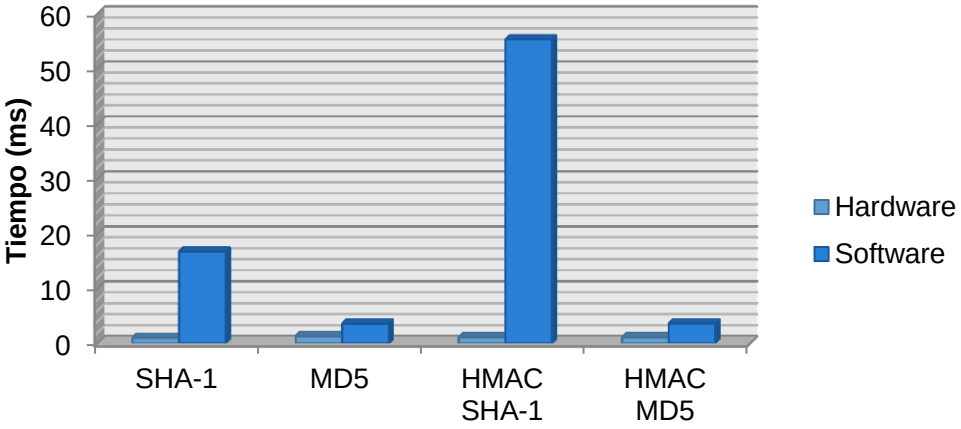


Figura 81: Consumo de algoritmos criptográficos simétricos

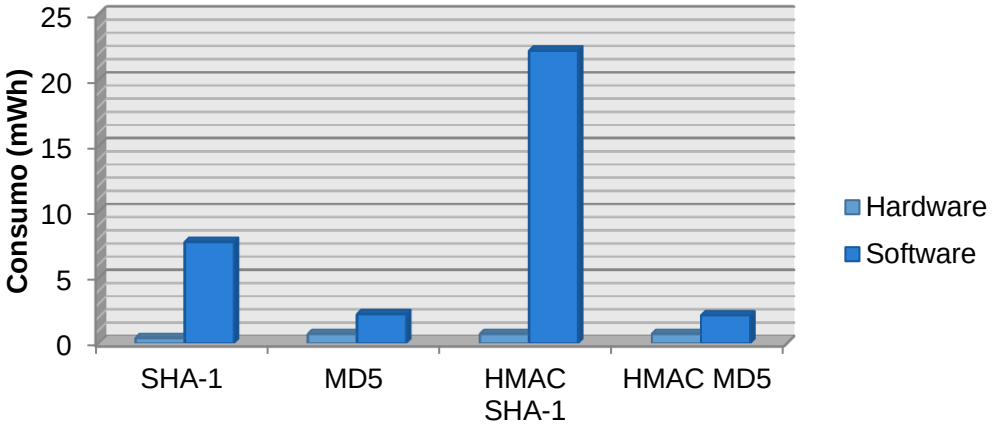
Para estudiar las prestaciones los algoritmos de autenticación, se han analizado los algoritmos SHA-1, MD5, HMAC-SHA-1 y HMAC-SD5. Los resultados obtenidos se muestran en la Figura 82. El tamaño del texto plano utilizado es el mismo que el usado para la comprobación de los algoritmos criptográficos. En este caso, la

relación entre los tiempos de ejecución de los métodos hardware y software es variable. El algoritmo MD5 es únicamente 4 veces peor en software que en hardware. Sin embargo, el consumo del SHA-1 en software es 17 veces más lento que en hardware. Las versiones HMAC muestran relaciones muy dispares. Mientras que el algoritmo HMAC-MD5 en software tiene un consumo 3 veces mayor comparado con la implementación hardware, el consumo del HMAC-SHA-1 en software es 20 veces peor que en hardware.

También se puede observar como la implementación software del SHA-1 requiere más potencia que la implementación software del AES. Sin embargo, el AES implementado en hardware requiere más potencia que el SHA-1.



(a) Consumo de tiempo



(b) Consumo de potencia

Figura 82: Consumo de algoritmos HASH

De acuerdo con los resultados obtenidos, podemos seleccionar los métodos óptimos para realizar el arranque seguro. Los algoritmos seleccionados son el HMAC-SHA1 para la autenticación y los AES-128 y AES-256 para el cifrado. En todos los casos se utilizará la implementación hardware. La elección del algoritmo HMAC-SHA1 se justifica porque según la Figura 80 es el más seguro de todos y sus tiempos y consumo son similares a los otros algoritmos de autentificación. En el caso de los algoritmos de cifrado, se han seleccionado los AES ya que, como se muestra en las figuras anteriores, su seguridad es mucho mayor que TDES y, además, su consumo es menor.

5.2.3. Proceso propuesto de arranque del sistema

A la hora de diseñar un procedimiento de arranque, una de las primeras tareas es determinar la memoria desde la cual se obtendrá el firmware que se desea cargar en el sistema. Lo ideal sería poder guardar el firmware en una memoria segura, a la que los potenciales atacantes no tuviesen acceso. Sin embargo, en la mayoría de los casos, las memorias que se pueden considerar seguras o cuyo acceso es difícil para un atacante (como por ejemplo las memorias que se encuentran integradas dentro de un chip) tienen una capacidad de almacenamiento muy limitada. Además, es bastante común que los nodos almacenen diferentes versiones de firmware en una misma memoria, para que el sistema se pueda adaptar a distintas necesidades o sea capaz de corregir su funcionamiento. Esto hace que se requiera más memoria que la necesaria para almacenar una única versión de firmware. Por último comentar que es frecuente que el tamaño de una única versión de firmware sea mayor que el espacio disponible en la memoria *flash* interna del dispositivo. Teniendo en cuenta todas estas limitaciones, muchos sistemas utilizan memorias *flash* externas (ej. tarjeta de memoria) para almacenar el firmware del nodo.

En la Figura 83 se puede ver un ejemplo típico de arquitectura HW de un nodo de una red de sensores inalámbrica. El nodo incluye un Sistema-en-Chip, STM32-F4, que integra en un solo dispositivo un procesador ARM, multitud de periféricos y módulos hardware para seguridad así como una memoria RAM y una *flash* de únicamente 1 Mbyte. En este trabajo se asume que todo lo que se encuentre almacenado dentro del dispositivo está seguro y no puede ser modificado por un atacante. Sin embargo, debido a la baja capacidad de la memoria interna, se hace indispensable el uso de una memoria *flash* externa (como por ejemplo una tarjeta de memoria, *SD-Card*) donde se almacena el firmware que se deberá cargar. Esta memoria la consideraremos insegura ya que cualquier atacante que tenga acceso al nodo podría manipularla o incluso cambiarla por otra.

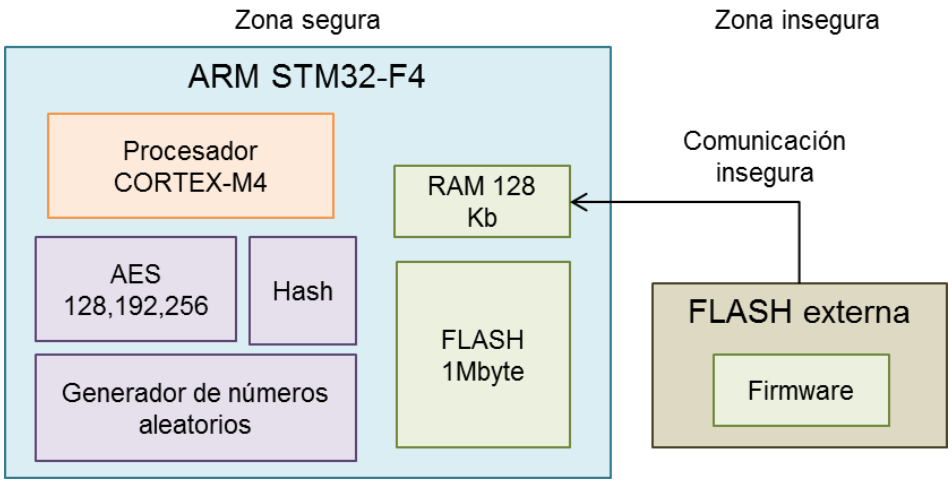


Figura 83: Zona segura e insegura del nodo

A continuación debe adaptarse el esquema típico de arranque del nodo, que fue presentado en el estado del arte, a las características propias de una red de sensores inalámbrica que integra nodos basados en dispositivos de tipo STM32F4. Una primera aproximación se presenta en la Figura 84, la cual añade al proceso presentado en la Figura 79 una nueva fase, para asegurar la autenticación y/o la confidencialidad del firmware que se va a cargar en el sistema.

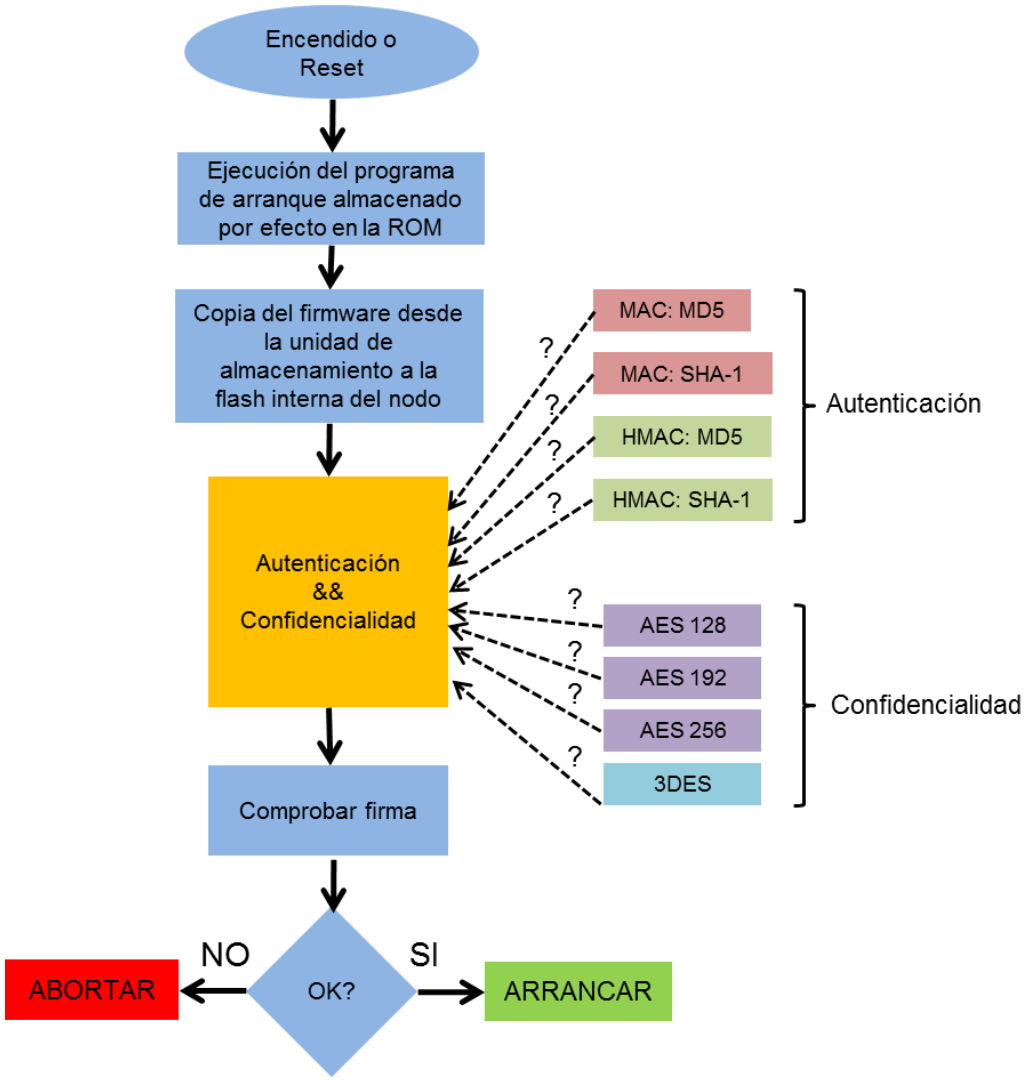


Figura 84: Diferentes posibilidades criptográficas en el arranque seguro

Aunque en la sección anterior se caracterizaron los métodos de encriptación y autentificación de forma aislada, es necesario analizar su comportamiento cuando se integran en el proceso de arranque. Para ello, se ha desarrollado e integrado una extensión del arranque genérico (Figura 85) en la plataforma Silica Xynergy-M4 [128]. El firmare tendrá que ser cargado desde una memoria *flash* externa (*SD-Card*), requiriendo los siguientes pasos:

- **Lectura desde la tarjera de memoria:**
  - En el caso de la plataforma seleccionada y por limitaciones de su arquitectura hardware, solo se podrá acceder a la tarjeta de memoria

(SD *card*) en bloques de 512 bits. Como consecuencia de esta limitación, cada paso de la Figura 85 trabajará con bloques de datos de 512 bits como máximo.

- **Confirmación de la autenticidad y/o desencriptado del código:**
  - Si es necesario realizar una autenticación, el firmware leído deberá tener incluida una firma. En este caso, el arranque tendrá que calcular el hash del firmware y compararlo con la firma recibida. Si ambas firmas son iguales podremos afirmar que el firmware es auténtico. En caso que se requiera validar la confidencialidad del firmware, el código estará encriptado y tendrá que ser desencriptado antes de ser usado.

- **Traducción de firmware:**
  - En algunas ocasiones, el firmware almacenado en la tarjeta de memoria no suele tener el formato adecuado para ser cargado en la memoria. Por ejemplo, en muchos sistemas el firmware se almacena en la memoria externa en el mismo formato que genera el compilador del sistema de desarrollo software (SDK): típicamente un fichero con formato hexadecimal (.hex). El uso de ficheros con dicho formato evita problemas relacionados con la ordenación de bits (*endianness*) y reduce el tamaño de la actualización.

- **Escritura del firmware traducido en la *flash*:**
  - Una vez que se ha transformado el firmware, se almacena el código resultante en la memoria *flash* del dispositivo. En primer lugar es necesario inicializar la memoria para permitir su posterior escritura. A continuación se escribe el firmware en la memoria *flash*. Debido a que este proceso puede consumir bastante tiempo, únicamente serán inicializados los sectores en los que se va a escribir.

- **Arranque de la aplicación:**
  - Una vez que todo el proceso de lectura, comprobaciones y escritura ha finalizado, el proceso de arranque transmitirá el control al nuevo

firmware. Para ello, es necesario que el procedimiento de arranque ceda el control a la función *main* del nuevo firmware.

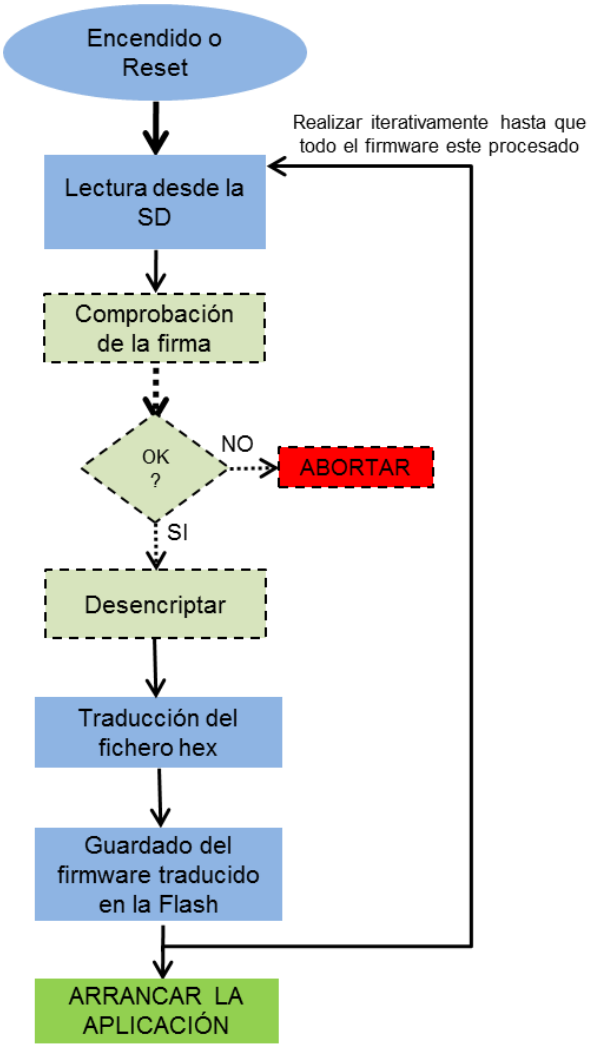


Figura 85: Diferentes posibilidades criptográficas en el arranque seguro

Una vez que se ha implementado el proceso de arranque genérico, es posible evaluar las prestaciones de cada algoritmo criptográfico dentro del proceso de arranque con total precisión. Para ello, se ha realizado una medición del consumo en la fuente de alimentación de la plataforma. Dicha medida se ha obtenido midiendo la caída de tensión en una resistencia con un osciloscopio Agilent MS09404A, trabajando a 4 Gsamples/seg [136]. Las medidas han sido tomadas con un *firmware* que tiene un tamaño de 170Kbytes y con el procesador de la plataforma trabajando a una frecuencia de 120MHz. La Figura 86 muestra el

consumo del proceso de arranque cuando el firmware es cifrado con un AES-256. Los resultados se han ampliado 20 veces para mejorar el análisis. El eje de abscisas es el tiempo en segundos y el eje de ordenadas representa la tensión, que es proporcional a la potencia consumida. En esta captura de pantalla del comportamiento del consumo del sistema se pueden identificar diferentes zonas: una primera etapa de arranque e inicialización de la plataforma; a continuación, el proceso de arranque y de procesado del firmware; por último, la fase en la cual se ejecuta la aplicación en modo normal. El consumo de energía total se calcula multiplicando la potencia por el tiempo total en el que el arranque está ejecutándose.

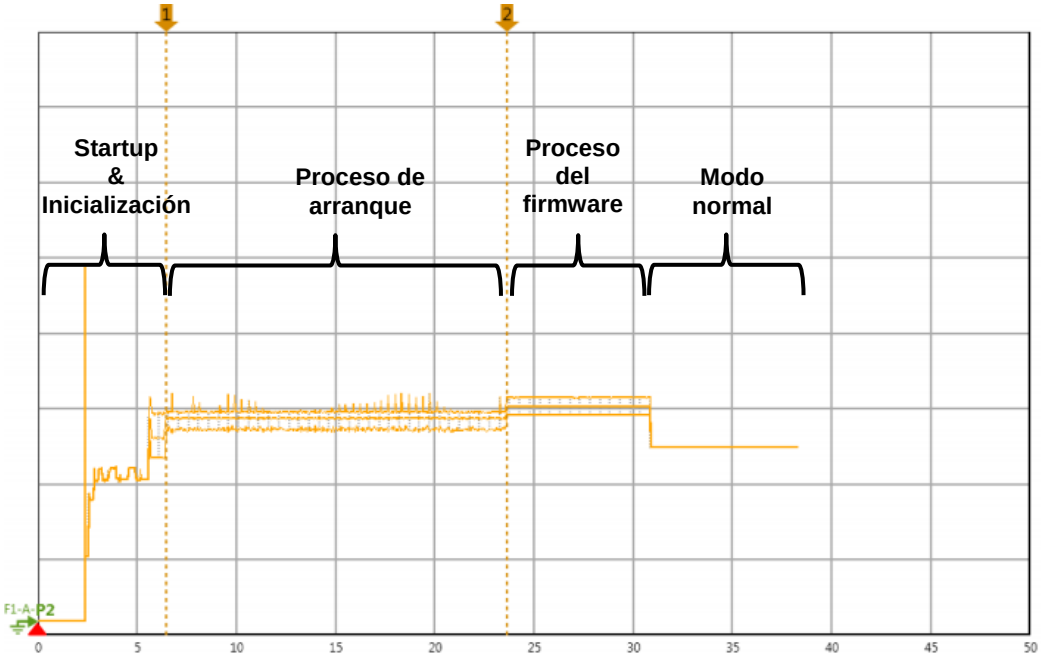


Figura 86: Ejemplo de medición de energía en el proceso de arranque

La Figura 87 muestra las medidas obtenidas para cada método criptográfico integrado en el proceso de arranque. Se puede apreciar el incremento del consumo comparándolo con el arranque sin seguridad. También se muestran las diferencias entre usar métodos hardware y software, lo que nos permite apreciar más claramente el impacto que tiene la selección de la solución hardware frente a implementaciones software en el arranque.

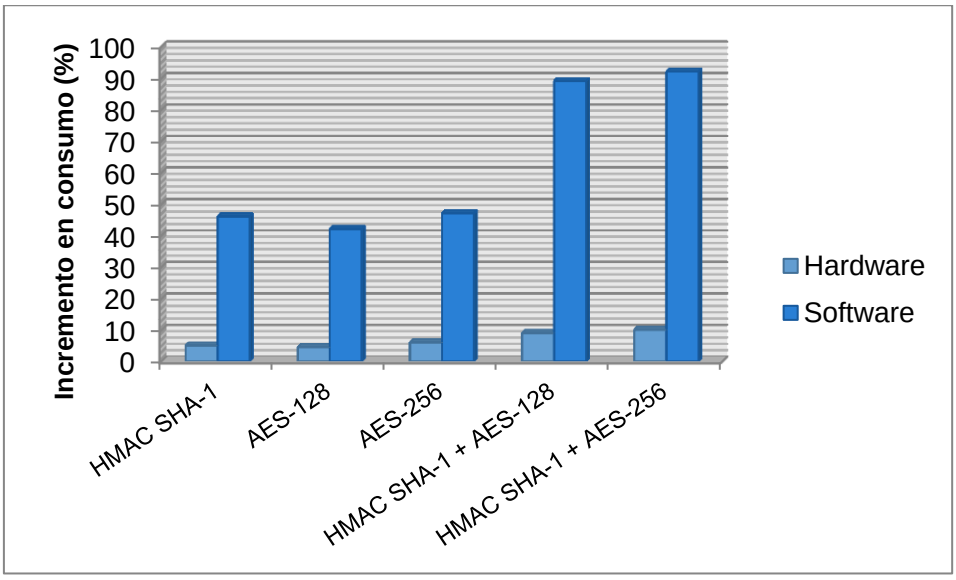


Figura 87: Incremento en porcentaje del consumo

Lo primero que podemos observar es, como se había comentado anteriormente, que el uso de métodos software incrementa demasiado el consumo. La implementación software de la combinación más segura (HMAC-SHA-1+ AES-256) muestra un incremento de casi el 100% comparado con la implementación hardware. Por lo tanto, lo recomendable seria descartar los métodos software en este tipo de sistemas. También se puede observar que la autenticación realizada con el HMAC-SHA-1 consume más que la encriptación con AES-128. Este resultado confirma los resultados vistos en la sección anterior.

5.2.3.1. Procedimiento de arranque flexible con diferentes niveles de seguridad

En esta sección, se va a proponer una estrategia de arranque seguro que incluyen una propiedad adicional: la posibilidad de establecer un procedimiento de inicio flexible que permita optar entre reducir el consumo o aumentar la seguridad en caso de que sea necesario. La razón principal que justifica esta aproximación es que cada nodo de una red no tiene los mismos requisitos de seguridad y consumo, ya que estos parámetros dependen de muchos factores, como por ejemplo su propósito o su zona de despliegue. Por lo tanto, para reducir el consumo, será

posible seleccionar el nivel de seguridad más adecuado para ese tipo de nodo. Por esta razón, será posible seleccionar un arranque sin seguridad o escoger un arranque que únicamente autentique el firmware, o que únicamente provea confidencialidad o ambas características conjuntamente.

Esta metodología tiene en cuenta la seguridad necesaria para evitar ataques manteniendo un bajo consumo y mejorar la mayor parte de aproximaciones estudiadas del estado del arte. Para ello, se han definido cuatro niveles de seguridad, como se ha comentado anteriormente:

- **Nivel 1:** Sin autenticación ni confidencialidad.
- **Nivel 2:** Creación de una firma con HMAC-SHA-1 (Autenticación).
- **Nivel 3:** Firmware cifrado con AES-256 (Confidencialidad).
- **Nivel 4:** HMAC-SHA-1 + AES-256 (Autenticación y Confidencialidad).

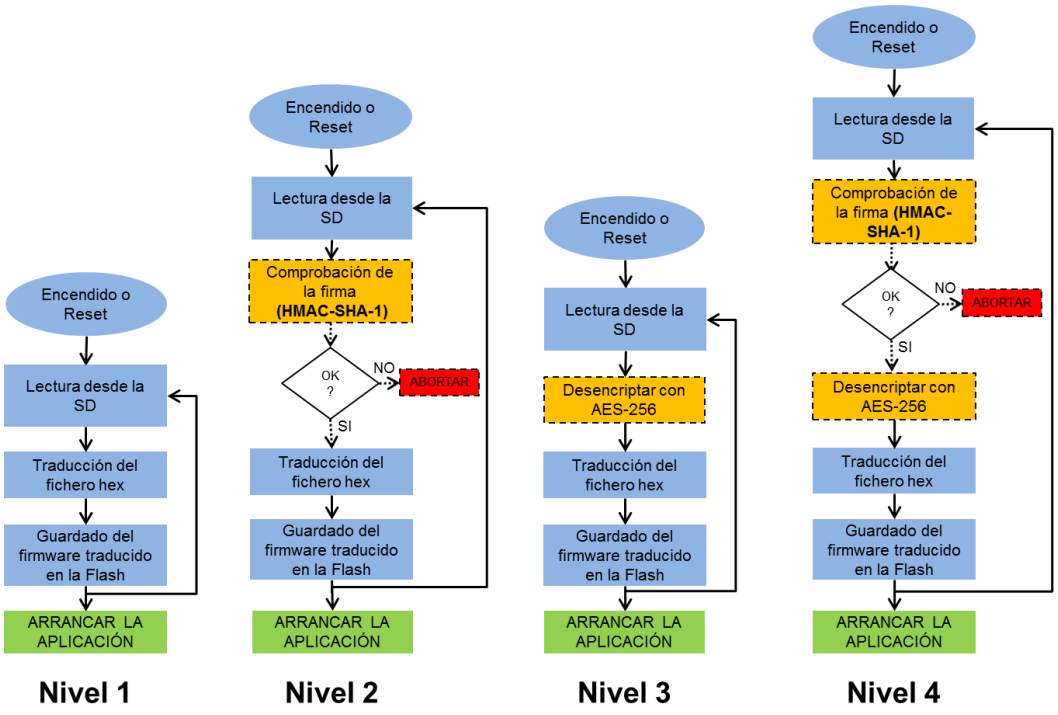


Figura 88: Esquema de cada nivel

En la Figura 88 se puede apreciar un esquema de la implementación de cada nivel. Podemos apreciar como el arranque del nivel 1 directamente copia el

firmware desde la tarjeta SD y, según lo va traduciendo al formato propio del nodo, lo va copiando en la memoria *flash* de destino. En cambio, en el nivel 2 (el cual soporta autenticación), la firma se va calculando según se lee el firmware para, finalmente, compararla con la incluida en el código, pasando a iniciar la aplicación si la comparación de ambas firmas es correcta. En el nivel 3, el proceso se inicia con una lectura y desencriptado del firmware. En caso de que el firmware estuviese encriptado con otra clave (o no fuese legítimo) el resultado no sería un firmware valido y no se podría cargar en la memoria *flash*. La validez del firmware en este caso se comprueba verificando marcas (números mágicos, *magic number*, o "marcas de agua") introducidas en el código. Finalmente, el último nivel y más seguro aplica ambas técnicas, tanto la firma para autenticar el firmware como el desencriptado para asegurar la confidencialidad.

Los niveles de seguridad están ordenados de menos seguro a más seguros. Como es esperable, el incremento de la seguridad lleva asociado un incremento en el consumo. Este incremento se puede observar en la Figura 89, donde se muestra el consumo de energía para cada nivel de seguridad. Con estas medidas, el diseñador podrá decidir en cada caso cuál es el nivel más apropiado para cada nodo o red.

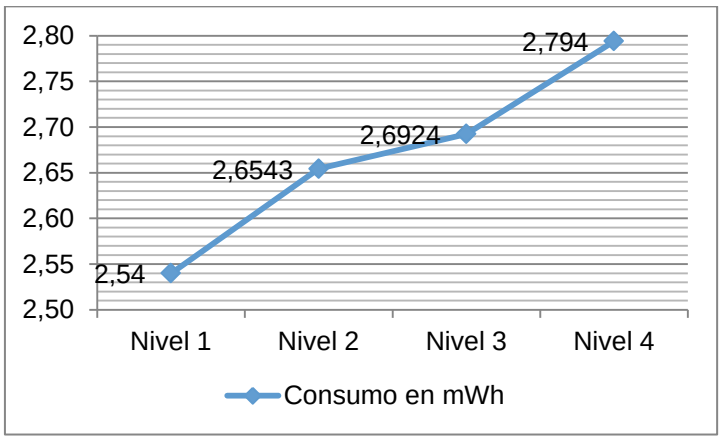


Figura 89: Consumo asociado a cada nivel de seguridad

Es importante comentar que no se ha incluido el AES-128 en ninguno de los niveles seleccionados. Esto es debido a que, según los datos experimentales presentados en secciones anteriores, el consumo del AES-256 es únicamente

alrededor de un 2% mayor en que el AES-128, por lo que su elección no compensa al ser la versión de 256 bits mucho más segura. Sin embargo, como ya se ha comentado, estos resultados están medidos en una plataforma específica, y aunque es esperable que los resultados sean similares en otras plataformas, los desarrolladores podrían replicar los pasos anteriormente presentados para asegurar que estos resultados concuerdan en la plataforma en la que se incluye el arranque flexible y seguro.

5.2.4. Validación del arranque flexible y seguro

Para comprobar el arranque implementado y validar los resultados de seguridad y consumo, se ha desarrollado una batería de test que serán aplicados a nodos reales. La estrategia de verificación utilizada evalúa cada nivel de seguridad con seis tipos de firmwares diferentes, cada uno de ellos con características específicas. El primer test está asociado a un firmware plano, que no incluye seguridad. El segundo test incluye un firmware corrupto (no funcional), que simula un firmware modificado o con alguna sección corrompida. El tercer tipo de test simula un firmware firmado correctamente. Los 3 tipos de test restantes han sido definidos para validar características específicas de cada nivel de seguridad. Por ejemplo, para validar el nivel 3 se ha desarrollado un test con un firmware malicioso que incluye una firma no válida o firmada con otra clave. Para validar el nivel 4 se han desarrollado test con firmwares que incluyen firmas y encriptaciones modificadas por atacantes.

La plataforma hardware intentará cargar en su memoria *flash* interna y arrancar el sistema con el firmware del test. El arranque propuesto deberá detectar los firmwares erróneos y/o maliciosos y abortar el proceso de arranque para proteger la integridad del sistema. Debido a que el consumo es muy importante en las redes de sensores inalámbricas, se ha medido el consumo en cada tipo de test, utilizando algoritmos HW y SW.

Tabla 25: Resultados de los test de arranque

Nivel de seguridad	Firmware	¿Carga correcta?	Consumo (Criptografía HW)	Consumo (Criptografía SW)
Nivel 1	Firmware sin encriptar	SI	2,54mWh	2,54mWh
	Firmware falso (atacado)	SI	0%	0%
	Firmware corrupto	NO	-6%	-6%
	Firmware firmado	NO	-87%	-87%
	Firmware encriptado	NO	-91%	-91%
	Firmware encriptado y firmado	NO	-91%	-91%
Nivel 2	Firmware sin encriptar	NO	-97%	-95%
	Firmware corrupto	NO	-97%	-95%
	Firmware firmado con clave correcta	SI	+4.5%	+46%
	Firmware firmado con clave incorrecta (atacado)	NO	-98%	-95%
	Firmware encriptado	NO	-98%	-95%
	Firmware encriptado y firmado	NO	-96%	-94%
Nivel 3	Firmware sin encriptar	NO	-98%	-95%
	Firmware corrupto	NO	-97%	-95%
	Firmware firmado	NO	-98%	-95%
	Firmware encriptado con clave correcta	SI	+6%	+47%
	Firmware encriptado con clave incorrecta (atacado)	NO	-98%	-95%
	Firmware encriptado y firmado	NO	-98%	-95%
Nivel 4	Firmware sin encriptar	NO	-97%	-95%
	Firmware corrupto	NO	-97%	-95%
	Firmware firmado	NO	-96%	-94%
	Firmware encriptado	NO	-97%	-95%
	Firmware encriptado y firmado con claves correctas	SI	+10%	+93%
	Firmware encriptado y firmado con claves incorrectas (atacado)	NO	-97%	-95%

La Tabla 25 muestra los resultados obtenidos con los test de verificación de la estrategia propuesta. En dicha tabla se ha utilizado como referencia el consumo de energía del proceso de arranque con nivel 1 y firmware plano. El resto de los consumos se representan como una variación del consumo de referencia.

Como se puede observar, en el nivel 1 (arranque no protegido) el ataque consigue prosperar y el arranque no es capaz de detectar el firmware modificado, por lo que este es cargado y ejecutado en la plataforma. Sin embargo, en los otros niveles (que implementan medidas de seguridad) los ataques no logran prosperar. Además, se puede observar que cuando un firmware no está configurado conforme al nivel de seguridad esperado, el código no llega a ser cargado. Por ejemplo, en el nivel 3, cuando se intenta cargar un firmware orientado al nivel 4, éste es descartado.

Con respecto a los casos en los que el firmware cargado es el esperado en el nivel de seguridad estudiado, se puede apreciar como el consumo aumenta cuando se utilizan métodos criptográficos basados en software, como era de esperar a la vista del estudio realizado anteriormente. Además, el impacto en consumo de la estrategia de arranque propuesta es limitada si se utilizan módulos hardware específicos. En esta línea se puede observar que en el caso de utilizar el nivel más seguro (nivel 4), donde se incluye la autenticación mediante firma y la confidencialidad mediante una encriptación por AES con una llave de 256 bits, el incremento del consumo es de únicamente un 10% comparado con el arranque sin seguridad.

6. Metodología de verificación de sistemas embebidos

A nuestro alrededor, los sistemas embebidos están constantemente creciendo, tanto en número como en complejidad. Ellos son responsables de nuestra seguridad, salud, cuestiones ambientales, etc. Además, estos sistemas deben proporcionar respuestas en tiempo real en entornos críticos, en donde se pueden causar daños a personas u otros seres y cosas. Debido al aumento en la complejidad de las funciones realizadas por este tipo de sistemas, así como a su comportamiento en tiempo real, la verificación se ha convertido en la tarea más costosa y compleja del proceso de diseño.

El proceso de verificación no solo tiene por objetivo la detección y prevención de errores, sino que también permite validar los requisitos funcionales y no-funcionales del sistema al tiempo que se garantiza que el diseño cumple con su propósito. Además, la metodología de verificación proporciona una garantía de calidad del software, que puede ser utilizada para obtener certificaciones específicas como, por ejemplo, la definida en el estándar de seguridad (safety) IEC 61508 [137].

La verificación del software embebido desde las primeras etapas del proceso de diseño permite un desarrollo más eficiente y reduce el tiempo de diseño y el coste del proceso. Durante el proceso de desarrollo de una aplicación software, el código

está continuamente modificándose y ampliándose, lo que aumenta enormemente la posibilidad de introducir errores o disfuncionalidades. Por esta razón, el plan de verificación define una estrategia que permite desarrollar pruebas (test) al mismo tiempo que se desarrolla el software. Las pruebas normalmente ayudan a detectar y encontrar los posibles problemas del código. Hay que destacar que el proceso más costoso no es arreglar un error o disfuncionalidad, sino entender dónde y porqué se produce el problema.

En general, los test o pruebas que se realizan al software deben cumplir una serie de requisitos básicos. En primer lugar, los test deben ser completos y abarcar el mayor porcentaje de código posible. En muchos casos se alcanza hasta el 100% del código. Para evaluar la amplitud del proceso de verificación se han definido diversas medidas de cobertura de test, como la cobertura de línea o salto. En segundo lugar, los test deben ser independientes, esto es, la ejecución de un test no puede afectar a otro test. En tercer lugar, la ejecución de los test debe ser rápida, para que afecte lo menos posible al tiempo de desarrollo. Hay que tener en cuenta que los test se ejecutan muchas veces (típicamente cada vez que se realiza una modificación o ampliación de código) por lo que se requiere que el tiempo de ejecución sea reducido. En cuarto lugar, la ejecución de los test debe ser automática evitando, en la medida de lo posible, cualquier intervención manual. Esto permite ejecutar grandes test de integración, por ejemplo, fuera del horario laboral de los diseñadores, lo que reduce su impacto en el proceso de diseño. Por último, los test deben ser reutilizables, es decir, deben poder ser utilizados durante todas las fases del proceso de diseño. Los test normalmente se definen a nivel sistema y el objetivo es poder reutilizarlos hasta en las pruebas en la plataforma física.

Aunque existen multitud de trabajos en la literatura orientados a la verificación de software, su aplicación al desarrollo de sistemas embebidos no es directa, debido a las especiales características de estos sistemas. La verificación de software en un sistema embebido es una tarea compleja que implica, además de la típica verificación funcional de la aplicación, la validación de requerimientos no-funcionales dependientes de la plataforma HW, como pueden ser tiempos de ejecución o consumo de energía. Por lo tanto, las herramientas de verificación de

software tradicionales tienen que ser complementadas con otras técnicas que verifiquen parámetros no-funcionales, como plataformas virtuales (simuladores) con capacidad de análisis de prestaciones o la propia ejecución del código en el hardware físico. El reto es poder reusar los test desarrollados durante el proceso de diseño en un ordenador en estos nuevos entornos (plataformas virtuales, hardware físico, etc.).

En esta sección se presenta una metodología que permite reusar los test durante todo el proceso de diseño. Dicha metodología combina técnicas clásicas de test de unidad de software (como GoogleTest [138]) con estrategias específicamente orientadas a la verificación de sistemas embebidos. El desarrollo de sistemas embebidos es un proceso de co-diseño HW/SW, en el cual la plataforma HW no está disponible durante la mayor parte del proceso. Debido a esto, no es posible validar los aspectos no-funcionales en las primeras etapas del desarrollo software. Por ello, en este trabajo se propone integrar el entorno de verificación con una plataforma virtual (por ejemplo VIPPE [139]) que permite emular y estimar las prestaciones del software cuando es ejecutado en la plataforma física, incluyendo el impacto de los RTOS que normalmente se utilizan en sistemas embebidos (por ejemplo FreeRTOS). Esto permite ejecutar los test en un ordenador (*host*) estándar que emula al hardware físico (*target platform*) al tiempo que se estiman y verifican parámetros no-funcionales (tiempo de ejecución, consumo, uso de la CPU, etc.) de la plataforma. El uso de plataformas virtuales también permite reducir el tiempo de test, ya que usa modelos de alto nivel de la plataforma hardware.

Por último, comentar que la metodología propuesta también permite reutilizar los test cuando la plataforma hardware esté disponible. Esta característica permite realizar simulaciones con "*Hardware-in-the-loop*", lo que incrementa la calidad y precisión de la verificación.

6.1. Técnicas de verificación: Estado del arte

Existe un gran interés, tanto en la industria como en la academia, en desarrollar metodologías que faciliten la verificación [140] del software durante el proceso de diseño. Por ejemplo, en [141] se evalúan una selección de entornos de test para los lenguajes C y C++, los mayoritariamente utilizados actualmente en sistemas embebidos. Específicamente, en el área de sistemas embebidos, existen trabajos centrados en técnicas que combinan diseño y test, como el desarrollo dirigido por test [142] [143] [144]. Igualmente, se describen una gran cantidad de herramientas y entornos de test en la literatura como, por ejemplo, [145], que presenta una amplia visión general de distintos entornos de test. Incluso los entornos de programación de código abierto como GNU proporcionan herramienta de análisis de la calidad del test como gcov, que proporciona medidas de cobertura clásica, como cobertura de sentencias, saltos, etc. [146]. En el área de test funcional, se han desarrollado entornos y métodos como [147] y [148]. Estos trabajos están centrados en el desarrollo de test unitarios o test que verifican cada módulo, clase o función (unidades) del software. El desarrollo de test de software está regulado por ciertas normas y, en esta línea, en [149] se propone una herramienta para realizar pruebas conforme al standard IEEE 1641. En el área de test de regresión existen trabajos, como el presentado en [150], donde los autores proponen una herramienta para realizar test automáticamente. En el área de test de parámetros no-funcionales, existen relativamente pocos trabajos en el estado del arte. Por ejemplo, [151] propone una herramienta para automatizar el proceso de las pruebas de fiabilidad para software embebido. Igualmente, existen multitud de trabajos orientados a verificación de aplicaciones específicas, como [152], en donde se presenta una herramienta para desarrollo de software de aplicaciones móviles. La verificación de la seguridad ha sido abordada por trabajos como [153], en donde se presenta una herramienta para desarrollar pruebas de seguridad centradas en la detección de fallos de seguridad. Igualmente se han desarrollado diversas metodologías cuyo objetivo es estandarizar el proceso de verificación como, por ejemplo, UVM (*Universal Verification Methodology*) [154], que está principalmente

orientada a la verificación de la parte hardware. En [155] los autores presentan una recopilación de las características de UVM y utilizan el lenguaje System Verilog [156].

En esta línea de definición de estándares para el proceso de verificación, se ha creado recientemente un grupo de trabajo cuyo objetivo es definir un lenguaje de especificación de test portables y estímulos que puedan ser usados para generar pruebas en distintas etapas del proceso de diseño. El "*Portable Stimulus Specification Working Group*" [157] creado por Accellera [158], señala que actualmente no hay una forma única para especificar los comportamientos de las plataformas destino (por ejemplo, emulación, hardware, simulación, etc.) que sean reutilizables. Para lograr este objetivo, el grupo trabaja en la definición de un lenguaje específico.

En el área de simulación del sistema, las tecnologías de simulación del software embebido dominantes son los simuladores del conjunto de instrucciones, ISS [159], la simulación nativa y la virtualización. La virtualización (como, por ejemplo, QEMU [160]) está basada en la ejecución en el *host* de binarios cross-compilados para los procesadores de la plataforma HW (*target*). Tanto la virtualización como la ejecución en un ISS producen simulaciones lentas, aunque el uso de un ISS puede proporcionar estimaciones de los tiempos de ejecución y consumos muy precisas. Por ser lentas, estas alternativas no son adecuadas para ejecutar los test rápidamente. La alternativa más eficiente en el estado del arte es la tecnología de simulación nativa [161]. Esta tecnología permite la construcción de plataformas virtuales con la suficiente precisión como para ser utilizadas en las primeras etapas del desarrollo. Típicamente, la simulación nativa, además de emular al sistema real, provee estimaciones rápidas y de alto nivel del tiempo de ejecución y consumo en la plataforma física de destino.

En este capítulo se presenta una metodología de verificación que permite reusar los test durante el proceso de diseño, abarcando varios niveles de abstracción. Dicha herramienta utiliza 2 entornos de verificación que serán presentados en la siguiente sección: un entorno para gestionar los test unitarios (GoogleTest) y un simulador virtual (VIPPE).

6.2. Entornos de simulación y test seleccionados

6.2.1. Entorno de test: GoogleTest

GoogleTest es el entorno para gestionar pruebas de unidades de código C++ utilizado por Google [138]. Éste es un entorno que soporta una gran variedad de plataformas (Linux, Mac OS X, Windows, Cygwin, Windows CE, y Symbian). Está basado en la arquitectura xUnit [162] y permite aplicar test unitarios a programas C/C++ con modificaciones mínimas del código fuente. Entre otras características, el entorno soporta el descubrimiento automático de pruebas, aserciones específicas y definidas por el usuario, diferenciación entre fallos fatales (que abortan la ejecución del programa) y no fatales, parametrización de los test por tipo y valores, capacidad de definir opciones para ejecutar los test y generación automática de los resultados del test (por ejemplo, puede indicar el número de test que abortan la ejecución, test ejecutados correctamente o pasados, etc.). Esta herramienta ha sido ampliamente utilizada en Google durante muchos años por cientos de sus desarrolladores. La estrategia de alto nivel de GoogleTest se presenta en la Figura 90. GoogleTest es el entorno encargado de introducir estímulos a un código que denominaremos FUT (*Function Under Test*). El FUT se ejecutará con dichos estímulos y devolverá al entorno de test los resultados generados. A continuación, Googletest se encargará de comprobar que los resultados generados son los resultados esperados. Para soportar esta estrategia, GoogleTest proporciona una serie de clases C++ que permiten identificar el FUT y gestionar sus valores retornados y estímulos (parámetros o datos de entrada del FUT).

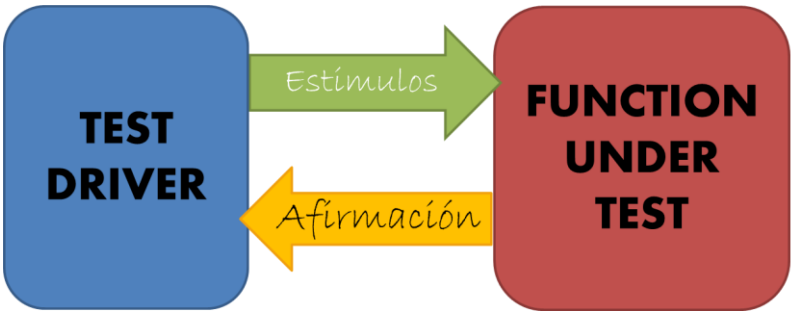


Figura 90: Funcionamiento de GoogleTest

6.2.2. Entorno de simulación: VIPPE

Como se ha comentado anteriormente, la simulación nativa es la tecnología que mejor combina la emulación del comportamiento y la estimación de las prestaciones del software embebido en la plataforma hardware con tiempos de simulación (o respuesta) reducidos. Por esta razón, el entorno desarrollado integra una plataforma virtual basada en simulación nativa para estimar las prestaciones antes de disponer de la plataforma física. En este contexto, se ha seleccionado la plataforma virtual VIPPE (*Virtual Parallel platform for Performance Estimation*) para simular las aplicaciones en las plataformas destino [139]. VIPPE es un entorno que usa métodos de instrumentación de código y compilación en el *host* para realizar la co-simulación HW/SW. Para reducir el tiempo de simulación, VIPPE combina una técnica de simulación paralela conservadora en el *host* con una estrategia de sincronización asíncrona de las tareas software que se ejecutan en el sistema. La simulación con VIPPE requiere una descripción de la plataforma HW/SW para crear la plataforma virtual donde el código de la aplicación será ejecutado. Esta aproximación permite a VIPPE modelar multitud de plataformas HW y RTOS basados en POSIX. Además puede utilizar información sobre el mapeo de los componentes de la aplicación en los elementos de la plataforma física.

6.3. Estrategia de verificación propuesta

Durante el proceso de diseño de un sistema embebido suele ser necesario realizar pruebas en diferentes entornos: plataforma virtual, plataforma física, test funcional en el entorno de desarrollo, etc. En esta tesis se ha desarrollado una metodología de test que identifica 4 estrategias de test:

1. Verificación funcional a nivel sistema. El objetivo es verificar el comportamiento sin tener en cuenta parámetros no-funcionales. Esta verificación se realiza en el *host* (computador en el que se desarrolla el sistema) utilizando GoogleTest para gestionar las pruebas. Esta es la estrategia que normalmente se utiliza en verificación de software no-embebido.
2. Verificación funcional y no-funcional en el *host*. En este caso, el entorno de test (GoogleTest) se combina con la plataforma virtual para verificar el sistema en el *host*. Por lo tanto, los test están en el *host*, mientras que el FUT se simula con la plataforma virtual.
3. Verificación en el *host* con hardware físico. En esta estrategia, los test se ejecutan con el entorno GoogleTest en el *host*, mientras que el código del FUT se encuentra en la plataforma física.
4. Verificación funcional y no-funcional en la plataforma física. En este caso, tanto los test (GoogleTest) como el código a verificar se ejecutan en la plataforma física.

La primera estrategia es la que normalmente se utiliza para verificar software con GoogleTest, por lo que no será comentada en esta sección. El resto de las estrategias se presentan en los siguientes apartados. La principal contribución de este capítulo es que los test definidos en la primera estrategia van a poder ser reutilizados en las siguientes.

6.3.1. Verificación funcional y no-funcional en el *host*

Esta estrategia permite verificar el software sin necesidad de disponer de hardware físico. Para ello se combina el entorno de verificación con una plataforma virtual que simula el comportamiento del sistema. El objetivo es que los test se gestionen utilizando GoogleTest, incluyendo la generación de parámetros y el análisis de los resultados. El código que está siendo verificado (FUT) se ejecuta en una plataforma virtual, lo que permite validar parámetros no-funcionales durante el test.

La Figura 91 muestra un esquema de la integración entre el entorno GoogleTest y la plataforma virtual VIPPE. En este caso, GoogleTest es el responsable de iniciar la ejecución de VIPPE, siendo ambos procesos ejecutados en el *host*. La comunicación entre ambos procesos (GoogleTest y VIPPE) se realiza mediante interfaces (*sockets*) del sistema operativo del *host*. El uso de *sockets* posibilita el futuro remplazo (sin apenas realizar cambios) del código de la plataforma virtual por la plataforma física cuando la misma esté disponible. Esto permite la reutilización de entornos de verificación con solo cambiar las librerías de soporte e interfaces de comunicación

El protocolo de operación básico se muestra en la Figura 92. GoogleTest usa las funciones definidas en la API de comunicación desarrollada para este propósito (Sección 6.4) para gestionar y comunicarse con la plataforma virtual. Como se puede observar en la Figura 92, es GoogleTest el que inicia la ejecución del simulador virtual (VIPPE), el cual espera recibir los argumentos de entrada para las funciones que tiene que ejecutar. Una vez que VIPPE tiene toda la información que GoogleTest le ha pasado a través de la comunicación vía *sockets*, GoogleTest da la orden de empezar la simulación (InitSim()). VIPPE será el encargado de ejecutar el código de usuario y estimar los parámetros no-funcionales. Una vez que termine la simulación, VIPPE enviará los parámetros funcionales y no-funcionales a GoogleTest, para que éste pueda hacer las comprobaciones oportunas. GoogleTest

comparará los resultados obtenidos con la simulación con los resultados esperados y determinará si el test se ha pasado o no.

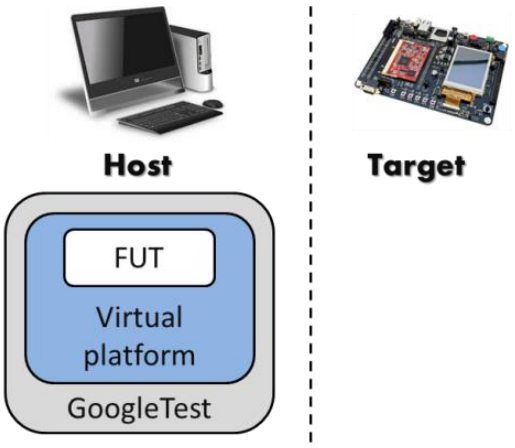


Figura 91: Ejecución de los test y el código en el *host* con una plataforma virtual

Esta aproximación permite la verificación de la funcionalidad y la estimación de los parámetros no-funcionales sin disponer de hardware físico. Por lo tanto, esta estrategia puede ser utilizada durante las primeras etapas del proceso de diseño.

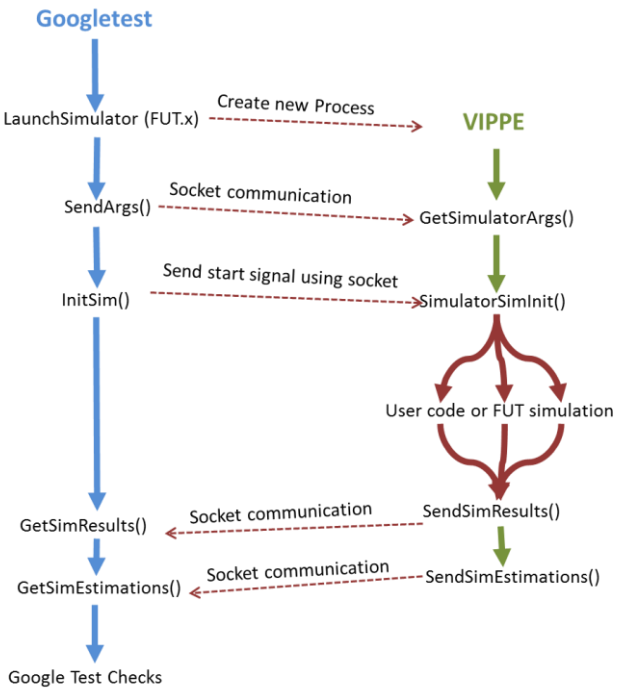


Figura 92: Interacción entre GoogleTest y VIPPE

6.3.2. Verificación en el *host* con hardware físico

En las etapas finales del proceso de desarrollo, a partir del momento en el que la plataforma hardware esté disponible, será posible utilizar la tercera estrategia propuesta. Para ello, se podrán definir o reutilizar los test realizados en las anteriores estrategias para verificar requerimientos tanto funcionales como no-funcionales. La Figura 93 muestra el esquema de cómo se asignan los procesos a cada entorno. Esta estrategia difiere de la presentada en la Figura 91 en que las funciones bajo test (FUT o código de la aplicación) son movidas desde la plataforma virtual a la plataforma hardware. Gracias a esto es posible obtener medidas no-funcionales con mayor precisión.

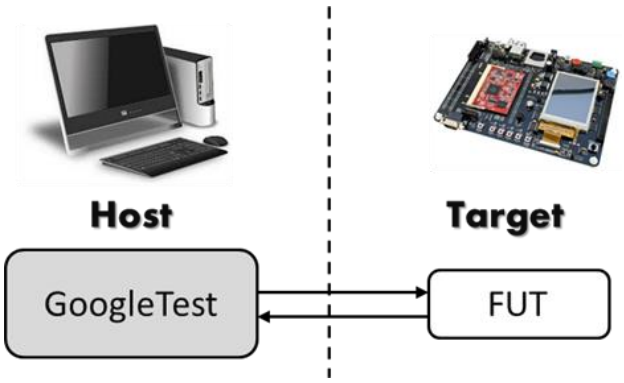


Figura 93: Ejecución de los test en el *host* y del código de usuario en el *target*

La comunicación entre GoogleTest (alojado en el *host*) y la función a testear (alojado en el *target*) se realiza utilizando *sockets*. El proceso de comunicación es similar al de la anterior estrategia, lo cual permite la utilización de los test utilizados con antelación. Esto permite acortar el tiempo de desarrollo, ya que no es necesario reescribir los test nuevamente con el cambio de estrategia, que permite tener el hardware en el "lazo de verificación" (*Hardware-in-the-loop*).

Para poder ejecutar la función bajo test en la plataforma hardware y monitorizar el comportamiento del sistema, es necesario introducir en el hardware un proceso ligero que controla el proceso de test. La idea principal es que este proceso no

altere (o su impacto sea el menor posible) el rendimiento del software sobre el sistema. Este proceso monitor (Figura 94) es el responsable de la comunicación entre la plataforma final y el *host* mediante el uso de *sockets*. En este caso, como se puede apreciar en la Figura 94, el monitor será el encargado de lanzar la FUT cuando GoogleTest lo requiera. Entonces la FUT se ejecutará en la plataforma hardware y enviará los resultados funcionales a GoogleTest. Los resultados no-funcionales, como tiempo o consumos, serán medidos por el monitor que será el encargado de trasmitírselos a GoogleTest. Como se ha comentado anteriormente, la comunicación se realiza usando funciones con el mismo nombre que en la estrategia anterior, pero enlazándolas con una librería específica de la plataforma. La Figura 94 muestra el diagrama del flujo de esta estrategia. El monitor y el FUT son compilados y ejecutados en la plataforma hardware. Por otra parte, el test se ejecuta en una maquina externa o *host*, para que no afecte al comportamiento de la plataforma HW. Como se puede apreciar comparando la Figura 94 y la Figura 92, el esquema de llamadas de GoogleTest es el mismo, lo que facilita la reutilización de test.

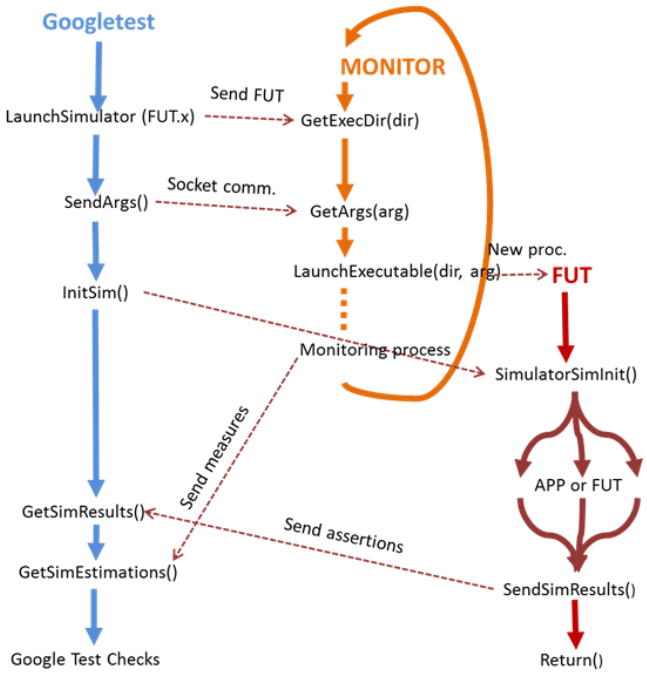


Figura 94: Interacción entre GoogleTest y la plataforma final

6.3.2.1. Utilización de *drivers* virtuales en verificación

La estrategia anteriormente comentada tiene una importante limitación: no permite interaccionar con el software de aplicación (FUT) durante la ejecución del test. Durante un test de software clásico, el FUT interactúa con el test en dos momentos: cuando recibe los parámetros para iniciar su ejecución (estímulos) y cuando retorna resultados después de su ejecución. Esta aproximación es válida para verificar funciones y clases, pero no es adecuada para verificar software embebido que, normalmente, está interactuando constantemente con el entorno. Para interactuar con el entorno, recibiendo estímulos y generando respuestas, el software utiliza servicios (*drivers*) del RTOS. Por lo tanto, durante la verificación de software embebido será necesario que el entorno de test gestione los valores que manejan los *drivers* del RTOS, permitiendo proporcionar estímulos y recibir resultados de la aplicación durante su ejecución en la plataforma física o virtual. En esta tesis se denomina "*driver* virtual" a una implementación de un *driver* de la plataforma hardware que es "gestionada" por el entorno de test. La Figura 95 muestra las 2 estrategias de verificación que usan *drivers* virtuales (la descrita en esta sección y la comentada en la anterior).

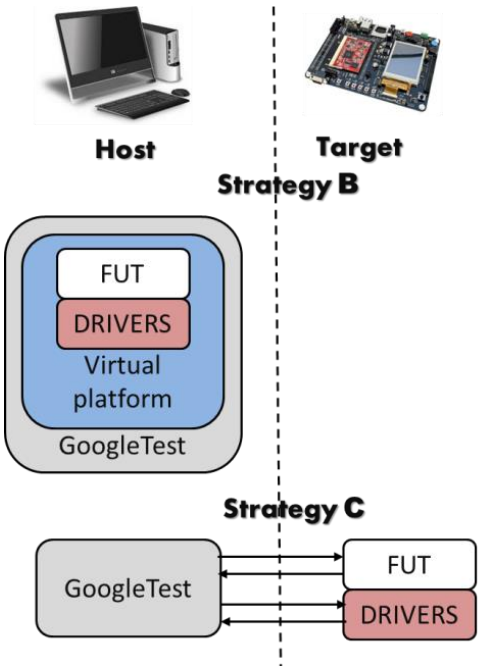


Figura 95: Estrategias presentadas con el uso de los *drivers*

Para poder crear *drivers* virtuales, se ha definido una interfaz muy simple, que será llamada desde el código de la aplicación o desde el modelo del RTOS. Cuando se quiera acceder a información externa (como podría ser, por ejemplo, leer la imagen de una cámara) se tendrán que utilizar cuatro funciones distintas: *OpenDevice*() y *CloseDevice*() para conectar y desconectar el dispositivo externo, *GetFromDevice*() para leer información del dispositivo y *WriteToDevice*() para enviar información al dispositivo. El test definido en GoogleTest será el encargado de determinar las acciones que deben tomarse y los valores que deben retornarse cuando una de estas cuatro funciones es llamadas en el FUT. Estas funciones podrán acceder directamente al periférico hardware asociado o, por el contrario, podrán proporcionar los datos que se declaren en el test.

Cuando se usan *drivers* virtuales en la plataforma virtual o física, la llamada al *driver* virtual implica un cambio en el tiempo de respuesta que podría afectar a la medida de los parámetros no-funcionales. Por ejemplo, supongamos que el *driver* que gestiona un sensor tarda 200 ms en retornar el valor leído. Sin embargo, cuando se ejecuta el *driver* virtual en la plataforma física el tiempo que tarda el *driver* en devolver el valor es diferente. Para minimizar este problema, las cuatro funciones anteriormente comentadas (*GetFromDevice*,…) incluyen como parámetro el tiempo de respuesta. Aunque en el caso de la plataforma virtual VIPPE, esta aproximación garantiza que la estimación de parámetros no-funcionales es correcta, cuando se utiliza en una plataforma hardware física, la especificación del tiempo de ejecución de los *driver* virtuales no siempre puede ser cumplida.

6.3.3. Verificación funcional y no-funcional en la plataforma física

La Figura 96 muestra un esquema de la cuarta estrategia propuesta para ejecutar las pruebas. En ella se implementa el entorno de test (GoogleTest) en la plataforma física (*target*). En este caso, GoogleTest es el responsable de ejecutar la

función bajo test (FUT) y extraer la información funcional y las medidas reales no-funcionales (tiempos de ejecución, consumos, etc.).

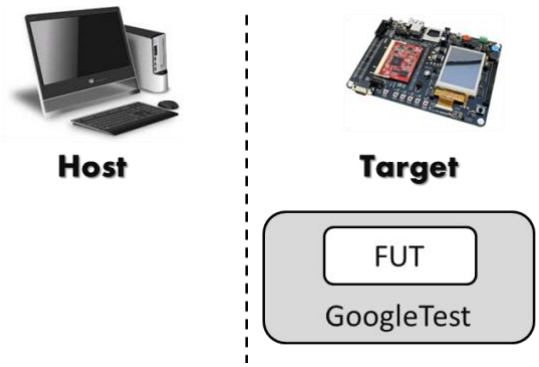


Figura 96: Ejecución de los test y el código en la plataforma destino

Esta estrategia puede plantear problemas en algunos sistemas embebidos, en donde los recursos disponibles (tamaño de memoria y/o pila, funcionalidad del RTOS, servicios disponibles en la plataforma, etc.) no sean suficientes para soportar la implementación de GoogleTest (por ejemplo, nodos de redes de sensores de bajo coste). Además, al ejecutarse el entorno de test concurrentemente con la aplicación que se está verificando, la media de parámetros no-funcionales (tiempo de ejecución, consumo) pueden verse afectadas, por lo que se generarían valores incorrectos. Aunque ésta es la estrategia que actualmente se usa en la verificación de sistemas embebidos, en la aproximación propuesta no es necesaria al estar totalmente superada por la anterior.

6.4. API de comunicación

Como se ha comentado anteriormente, se ha desarrollado una API que permite la comunicación entre el entorno de verificación y las plataformas física y virtual. El objetivo es definir unas funciones genéricas que permiten el uso de las distintas estrategias descritas anteriormente sin necesidad de modificar ni el código de los test ni la aplicación. La librería definirá diferentes funciones que permitirán comunicar los diferentes procesos descritos en las estrategias anteriores usando *sockets*. El API se ha dividido en 3 módulos básicos:

- TestEnvComAPI: Interfaz de comunicación del entorno de verificación (*Test-Environment-Communication-API)*. En este módulo se incluyen las funciones que se utilizarán en la los test que gestiona GoogleTest.

- PerfEnvComAPI: Interfaz de comunicación con el entorno de análisis de prestaciones y medida de parámetros no-funcionales (*Performance-Environment-Communication-API)*. Este módulo define las funciones que se utilizarán en la parte del simulador y de la plataforma *hardware*.

- TestEndDevices: Modulo en donde definen las funciones de acceso a los *drivers* virtuales explicados en la sección 6.3.2.1

Esta sección no pretende hacer una descripción completa de todas las funciones implementadas o presentar un manual de usuario de la herramienta de verificación, sino que busca proporcionar una visión global de la funcionalidad mínima necesaria para poder implementar la metodología propuesta en este capítulo.

6.4.1. Funciones de comunicación del entorno de verificación (TestEnvComAPI)

Este módulo incluye las funciones que podrán ser utilizadas en la parte de definición de los test. Algunas de estas funciones se incluirán en los ficheros propios de GoogleTest, como “test_usecase.cc”, para facilitar la gestión del test. Las principales funciones son:

- **int LaunchSimulator(char *const SimArgs[]):** Esta función inicia la ejecución del simulador VIPPE (en el caso de la estrategia que usa una plataforma virtual) o indica al monitor de la plataforma que un nuevo test va a ser lanzado (en el caso de la estrategia con la plataforma hardware).
- **int SendSimulationRawArgs(void *address, unsigned int nbytes):** Esta función envía los argumentos que se definen en GoogleTest al simulador o a la plataforma hardware utilizando *sockets*.

- **int InitSim(void):** Envía al simulador o al monitor de la plataforma hardware la orden de empezar la ejecución.
- **int GetSimRawResult(void *address,unsigned int nbytes):** Esta función solicita los resultados de la ejecución del FUT en la plataforma hardware o el simulador.
- **int GetSimEstimation(char *feature, P &v):** Mediante esta función GoogleTest puede solicitar parámetros no-funcionales medidos en la plataforma real o estimados con el simulador.
- **int WaitSimEnd(void):** Esta función detiene al test hasta que la simulación o la ejecución en la plataforma HW haya terminado, para poder solicitar los resultados del proceso.
- **int ShutDownSim(void):** Finaliza la simulación o indica al monitor de la plataforma real que ha terminado para cerrar la comunicación mediante *sockets*.
- **int InitDrivers(void):** Crea un nuevo hilo que espera peticiones para poder acceder al periférico a través del *driver* virtual.
- **int DeleteDrivers(void):** Elimina todos los canales de comunicación con los *drivers* virtuales, eliminado los hilos generados.
- **HWdevice* NewDevice(void):** Crea un nuevo *driver* virtual para un cierto dispositivo hardware.

6.4.2. Funciones de comunicación de parámetros no-funcionales (PerfEnvComAPI)

Las funciones definidas en esta sección se incluirán en el código de la FUT y tienen por objetivo comunicar parámetros no-funcionales (PerfEnvComAPI, *performance-environment-communication-API*) y comunicarse con los *drivers* físicos y/o virtuales. Estas funciones se llamarán en los ficheros propios del código que se verifica.

- **int GetSimulationRawArgs(void *address, unsigned int nbytes):** Recibe los argumentos de entrada enviados por GoogleTest (usando la función SentSimulationRawArg) para ejecutar el código.
- **int SimulatorSimInit(void):** Indica que ha recibido todos los argumentos necesarios y puede continuar la ejecución.
- **bool SendSimResult(Q &first, Args&... args):** Envía los resultados obtenidos a GoogleTest.
- **Device_t OpenDevice(char *id):** Abre un canal de comunicación con el dispositivo identificado con "id". La función envía una petición y espera a recibir una confirmación si el dispositivo existe.
- **int GetFromDevice(Device_t dev, void* dest):** Recibe los datos leídos del dispositivo . Se asume que el tiempo de ejecución del *driver* es cero. Ejecuta la función definida en el *driver*.
- **int GetFromDevice_timed(Device_t dev, void* dest):** Recibe los datos leídos del dispositivo . El simulador asume que el tiempo de ejecución de la funcionalidad del *driver* es el definido durante su especificación. Ejecuta la función definida en el *driver*.
- **int SendToDevice(Device_t dev, void* src, int size):** Envía información al dispositivo. Ejecuta la función de escritura definida en el *driver*.
- **int CloseDevice(int dev):** Cierra el canal de comunicación con el dispositivo identificado con "dev".

6.4.3. Funciones de configuración de los *drivers* (TestEndDevices)

Estas funciones son las encargadas de implementar la funcionalidad de los *drivers* virtuales en los test.

- **const char* ShowId(void):** Muestra el id del dispositivo.
- **int SetDeviceId(string idDev):** Da un id al dispositivo.

- **int SetUpDriverRead(int (*function)(void**)):** Establece la función de lectura del dispositivo que ejecutara el *driver* cuando sea llamado por las funciones **GetFromDevice** o **GetFromDevice_timed**.
- **int SetUpDriverWrite(int (*function)(void*,int)):** Establece la función de escritura que ejecutará el *driver* cuando sea llamado a través de la función **SendToDevice**.
- **bool CheckDriverRead(void):** Comprueba que la lectura ha sido correcta.
- **bool CheckDriverWrite(void):** Comprueba que la escritura ha sido correcta.
- **int SetTimeRead(long long int time):** Establece el tiempo de lectura del *driver* en nanosegundos.
- **long long int GetTimeRead():** Retorna el tiempo de lectura del *driver* en nanosegundos.

6.5. Base de datos de resultados

Para una correcta gestión de los resultados de los test se ha diseñado una base de datos en la que el entorno de verificación descarga automáticamente los resultados obtenidos en una ejecución del conjunto de pruebas. El entorno de verificación proporciona información sobre el resultado de cada test y los tiempos de ejecución. Adicionalmente, se ha incluido soporte para otras medidas, como la cobertura de código. Para ello, se han utilizado herramientas de análisis de prestaciones estándar en el *host* o plataforma física (entornos gcov y gprof de GNU).

La Figura 97 muestra el modelo de entidad-relación de la base de datos donde son almacenados los resultados de los test.

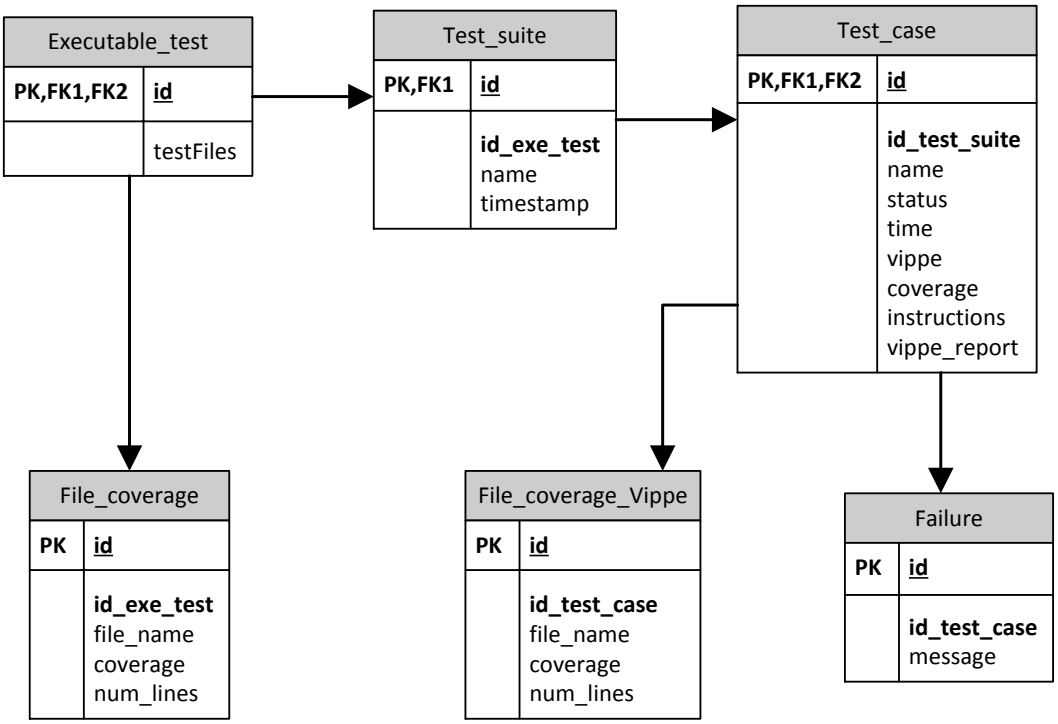
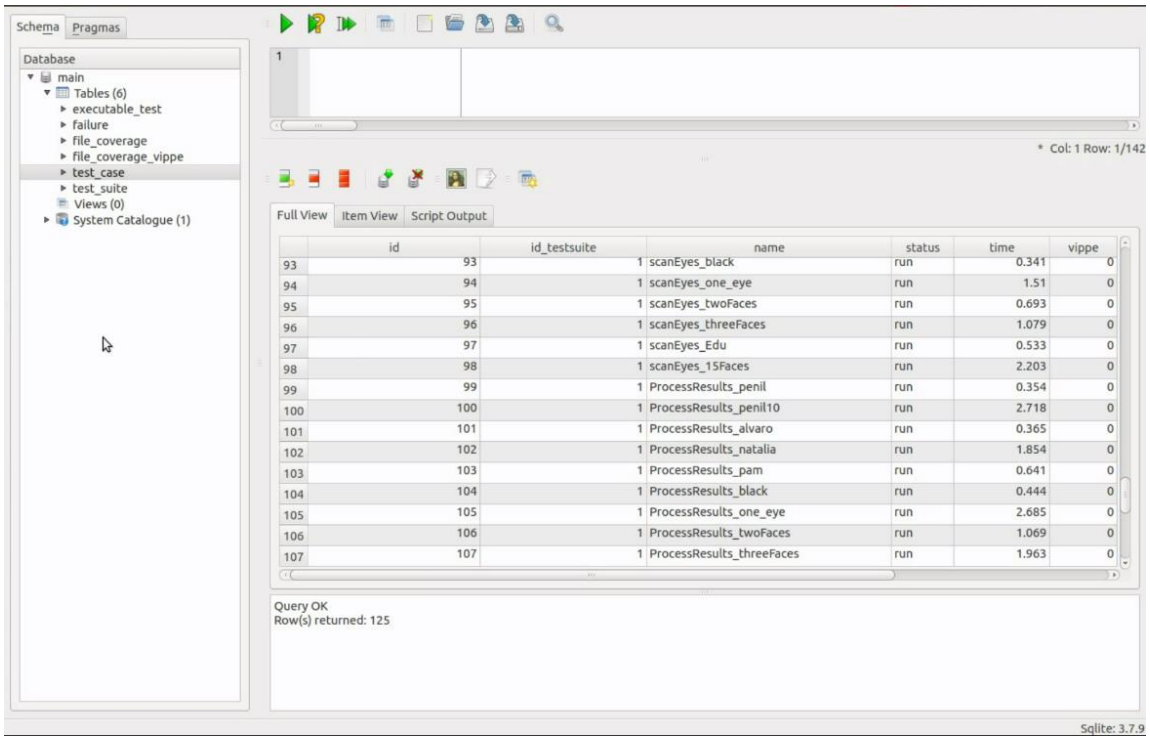


Figura 97: Modelo E-R de la base de datos de los reportes de validación

La ejecución de los test genera automáticamente y realiza las consultas para insertar los informes en la base de datos. Estas consultas son insertadas mediante transacciones. Las consultas insertan diferentes datos dependiendo del tipo de estrategia de verificación que se haya utilizado. Por ejemplo, las tablas “Executable_test”, “Test_suite”, “Test_case”, y “Failure” son tablas comunes para todas las estrategias. Sin embargo, la tabla “File_coverage” se rellena únicamente cuando la ejecución no se realiza en una plataforma virtual que no genere medidas de cobertura.

Esta base de datos puede ser consultada por cualquier programa de gestión de base de datos como puede ser SQlite [163]. En la Figura 98 se puede apreciar un ejemplo de una consulta a la base de datos mediante el programa SQlite.



	id	id_testsuite	name	status	time	vippe
93	93	1	scanEyes_black	run	0.341	0
94	94	1	scanEyes_one_eye	run	1.51	0
95	95	1	scanEyes_twoFaces	run	0.693	0
96	96	1	scanEyes_threeFaces	run	1.079	0
97	97	1	scanEyes_Edu	run	0.533	0
98	98	1	scanEyes_15Faces	run	2.203	0
99	99	1	ProcessResults_penil	run	0.354	0
100	100	1	ProcessResults_penil10	run	2.718	0
101	101	1	ProcessResults_alvaro	run	0.365	0
102	102	1	ProcessResults_natalia	run	1.854	0
103	103	1	ProcessResults_pam	run	0.641	0
104	104	1	ProcessResults_black	run	0.444	0
105	105	1	ProcessResults_one_eye	run	2.685	0
106	106	1	ProcessResults_twoFaces	run	1.069	0
107	107	1	ProcessResults_threeFaces	run	1.963	0

Figura 98: Consulta a la base de datos con SQlite

6.6. Caso de estudio: Reconocedor de caras

Para evaluar la metodología propuesta, se ha implementado y verificado con la misma una aplicación de tiempo real de procesado de video. Esta aplicación reconoce personas. Para ello incluye un detector de caras y ojos conjuntamente con un reconocedor de caras.

La Figura 99 muestra el esquema del algoritmo implementado. Se puede ver como el programa obtiene una imagen a través de la cámara, la preprocesa para poder trabajar con ella más ágilmente y realiza la detección de ojos y caras. Una vez determinados los ojos y caras que se identifican en la imagen, utiliza un clasificador en cascada. Finalmente, se ejecuta el reconocedor de caras que hace una predicción sobre quiénes son las personas capturadas.

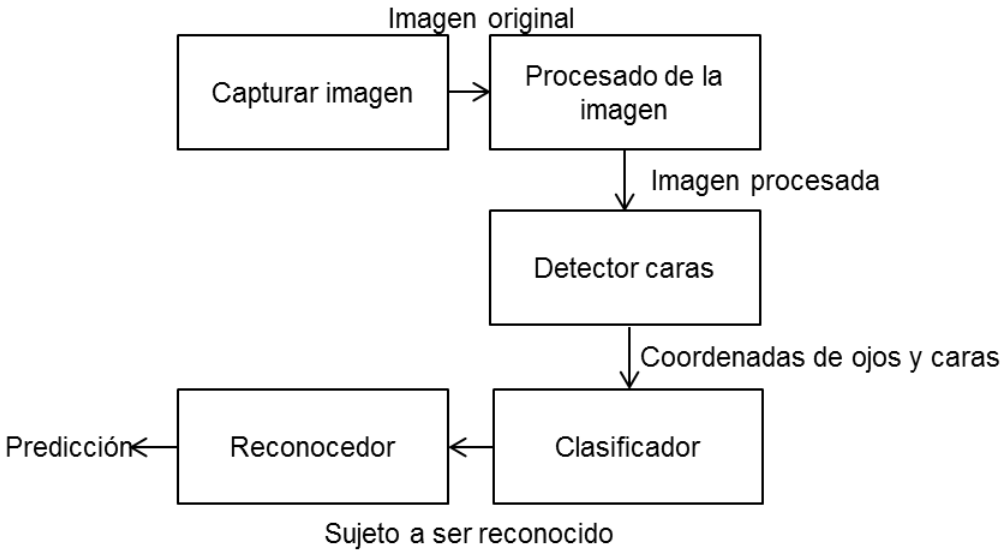


Figura 99: Esquema principal de la aplicación de reconocedora de caras

La verificación de este sistema se ha realizado utilizando las estrategias presentadas en este capítulo. Durante la fase inicial del proyecto, se planeó la utilización de una plataforma con un único procesador, pero gracias a las estimaciones obtenidas con el uso de la estrategia de verificación mediante la plataforma virtual, se pudo comprobar que el rendimiento de la aplicación era demasiado bajo y que la tasa de imágenes que se podían computar en tiempo real era muy baja. Para solventar este problema, se decidió paralelizar el algoritmo y utilizar una plataforma hardware con múltiples procesadores, de manera que cada núcleo se centrase en una tarea. Teniendo en cuenta los requisitos y placas hardware disponibles, se seleccionó la plataforma ODROID-XU+E [164] para implementar el sistema. Esta plataforma tiene una arquitectura basada en ARM con un procesador Samsung Exynos5 con cuatro procesadores Cortex A15 y cuatro procesadores Cortex A7. Esta plataforma es muy interesante para el desarrollo de software embebido crítico ya que tiene integrados sensores que permiten medir el consumo de potencia en tiempo real. Esto nos ha permitido añadir condiciones de consumo en la verificación con la plataforma hardware. El cambio de plataforma (pasando de un núcleo a ocho) no ha supuesto ningún problema para la verificación con la plataforma virtual, ya que solo es necesario redefinirla en el simulador y ejecutar exactamente el mismo código obteniendo con ello estimaciones de parámetros no-funcionales como tiempos de ejecución o consumo de energía.

Tabla 26: Aproximación del número de test pasados

Test	Tests cases	Test	# de test	# de Aserciones	# total de test	Aserciones totales.
Unit	Utils	writeImage*	30	≈ 900.000	1605	Más de 1.000.000 de aserciones Alrededor de 3000 aserciones no funcionales
		group rectangles_*	25	77		
		…	**…**	**…**		
		clipLine_*	43	89		
	InitScale	Init_*	2	≈30.000		
		Multiple_*	20	35		
	…	**…**	**…**	**…**		
	Init recogniz	Init_*	5	10		
		Null_Arg_*	2	4		
		…	…	…		
		BadPath_*	2	2		
Integration	ReadIma	NoCam_*	50	≈1.000.000	700	Más de 2.000.000 de aserciones 1400 aserciones no funcionales
		…	**…**	**…**		
		Cam_*	25	50		
		Color*	15	≈ 600.000		
	…	**…**	**…**	**…**		
	Process	Alvaro_*	5	12		
		…	**…**	**…**		
		Pablo_*	7	14		
Acceptation	Main	TwoFaces_*	23	60	387	1254 Aserciones 752 Aserciones no funcionales
		…	**…**	**…**		
		CallMain_*	50	120		
	ExPhoto	Photos_*	150	150		
	ExeCam	Camera_*	15	45		
Accuracy	Output	1Face_*	1000	1000	1300	3712 aserciones
		…	**…**	**…**		
		Glasses_*	93	219		
				TOTAL	3992	Más de 3.000.000 de aserciones

El conjunto de pruebas desarrollado incluye test de aceptación (pruebas muy amplias que buscan verificar la aplicación completa), test de integración (test medianos que busca verificar los componentes de manera aislada) y test unitarios (test pequeños que verifican funcionalidad de pequeñas partes del código o unidades). Además, se han desarrollado test de validación del algoritmo implementado que miden, por ejemplo, el número de detecciones correctas que realiza el reconocedor.

Los test de aceptación verifican el correcto funcionamiento de toda la aplicación. Los diferentes test desarrollados utilizan diferentes imágenes que la aplicación recibe como entrada. Usando el soporte para *driver*s virtuales presentado en la sección 6.3.2.1, el FUT accede a la imagen y genera un resultado que es analizado por el test. El FUT genera distintos parámetros como resultado, tales como el número de caras y ojos detectados o el sujeto reconocido.

Los test de integración validan cada componente de la aplicación. Por ejemplo, se ha implementado una batería de test para validar la fase de procesado de la imagen. Cada test estimula la clase que se está probando con una imagen distinta y el test comprueba que el procesado de esa imagen es correcto.

Los test unitarios validan las unidades de código. Por ejemplo, la parte del clasificador de la aplicación está compuesta por diferentes elementos como un buscador de ojos y caras. Los test unitarios comprueban cada una de los elementos que conforman el módulo del clasificador.

Finalmente, los test de precisión nos han ayudado a comprobar la eficacia del programa, ya que permite evaluar cómo afectan los cambios en el algoritmo implementado y la precisión en el reconocimiento de caras.

Como se muestra en la Tabla 26, se han implementado alrededor de unos 4000 test diferentes que incluyen, test unitarios, de integración, de aceptación y de precisión. Dichos test obligan a realizar más de 3.000.000 de comprobaciones funcionales y más de 5000 comprobaciones de parámetros no-funcionales. Por ejemplo, se han implementado más de 1600 test unitarios con más de 1.000.000 de validaciones. Con estos test se ha obtenido una cobertura de sentencia cercana al

95%. Los tiempos de ejecución de los test se presentan en la Tabla 27. El tiempo para pasar los test unitarios es de 735 segundos, para la ejecución mediante la plataforma virtual, y de 915 segundos, para la ejecución directa en la plataforma hardware. Con el objetivo de medir la tasa de acierto del reconocedor, se han utilizado los test de precisión que comprueban el número de caras, ojos y la persona reconocida. Es importante remarcar que el tiempo empleado en desarrollar los test ha sido inferior al 15% del tiempo del proceso de desarrollo. Esta reducción es consecuencia de la metodología utilizada. Además, se puede afirmar que, gracias a los test, se pudieron detectar muchos errores.

En resumen se ha estimado que el tiempo de desarrollo, en este caso de uso, se ha reducido alrededor de un 40% gracias al uso de la metodología, que detectó en las primeras etapas del desarrollo que la paralización del código no era adecuada. Además, se estima que el esfuerzo de corrección de errores ha sido reducido alrededor de un 75%, ya que la mayoría de los errores habían sido detectados y corregidos durante las primeras fases de desarrollo del software.

Tabla 27: Tiempos de ejecución de los test

Test	Tiempo de ejecución usando la estrategia con la plataforma VIRTUAL	Tiempo de ejecución usando la estrategia con la plataforma REAL
Unit tests	735 s	915 s
Integration tests	991 s	1200 s
Acceptation tests	597 s	1345 s
Accuracy tests	1013 s	3086 s

7. Conclusiones

En esta tesis se han presentado un conjunto de técnicas de análisis de sistemas embebidos en red, principalmente orientadas a la simulación de redes de sensores inalámbricas. Este trabajo ha sido ampliado para permitir el análisis del comportamiento de redes que están siendo atacadas, utilizando dicho análisis para guiar el desarrollo de software seguro, capaz de detectar y minimizar el impacto de los ataques a la red. También se han desarrollado técnicas que mejoran aspectos del sistema, tales como el arranque seguro o la actualización eficiente del software en campo. Con objeto de analizar la seguridad de la red, se ha desarrollado e integrado en el simulador una medida que permite estimar la seguridad de las comunicaciones que se establecen entre nodos de la red. Por último, el desarrollo se ha completado con un entorno que automatiza y facilita el desarrollo de test del sistema. Como resultado de este trabajo, se ha desarrollado un entorno completo de simulación y análisis de sistemas embebidos en red seguros que ha sido evaluado y comparado con diversos ejemplos reales. A continuación, se detallan los objetivos por capítulo, destacando las contribuciones más importantes del autor de la tesis en cada uno de ellos.

En el capítulo 2, se ha descrito la metodología de simulación de redes de sensores inalámbricas. A partir de una técnica de simulación HW/SW en la que trabaja el grupo de investigación (sección 2.3.1), el autor de la tesis ha desarrollado e integrado un modelo de red inalámbrica (sección 2.3.3) y de componentes

específicos de WSN (sección 2.3.4), además de integrar uno de los RTOS más utilizados en este tipo de redes, FreeRTOS (sección 2.3.2). El autor de la tesis también ha participado en la mejora de la metodología de simulación (sección 2.3.1) y en la presentación de resultados (sección 2.3.5). La simulación es muy rápida y permite simular cientos de nodos en un solo ordenador en segundos. Además, los errores de las estimaciones son pequeños (entre el 5% y el 8%), como muestra la sección 2.4.5.

A continuación, se presenta en el capítulo 3 un entorno de simulación de ataques totalmente desarrollado por el autor de la tesis. Partiendo de un estudio de los tipos más frecuentes de ataques a redes de sensores (sección 3.3), se han desarrollado e integrado en el simulador tres modelos de atacantes (secciones 3.4, 3.5 y 3.6) que permiten simular la mayoría de los ataques frecuentes identificados. Gracias a esta simulación, es posible estimar el impacto de los ataques, observando en los ejemplos del caso de uso (sección 3.7) incrementos del consumo de hasta un 150%, lo que podría poner en riesgo el ciclo de vida del nodo por agotamiento prematuro de su batería. A partir de este análisis, se ha descrito en la sección 3.8 una metodología que permite desarrollar firmware (software que se ejecuta en el nodo) capaz de detectar y minimizar el impacto de los ataques.

El capítulo 4 describe una medida de la seguridad de los mensajes que se transmiten por una red de comunicaciones. Dicha medida está basada en el cálculo de la entropía del mensaje y permite una implementación óptima en el simulador de redes de sensores inalámbricas. En este capítulo, el autor de la tesis ha contribuido con más del 50% del trabajo realizado, siendo el principal investigador en esa área.

Para mejorar la seguridad de los sistemas en red, no solo es necesario modificar el software que se ejecuta en los nodos, sino también es preciso disponer de un RTOS que ofrezca servicios seguros y eficientes. En el capítulo 5 se han presentado dos técnicas que permiten mejorar la seguridad del nodo (arranque seguro y actualización parcial del software) y que han sido desarrolladas totalmente por el autor de la tesis.

Por último, en el capítulo 6 se presenta un entorno que facilita y automatiza la gestión de pruebas. Dicho sistema integra una librería de código abierto de test de

unidad (GoogleTest) y facilita la portabilidad de test (o pruebas) entre diferentes niveles de abstracción (sistema, plataforma virtual y sistema/tarjeta física). El entorno ha sido totalmente desarrollado por el autor de la tesis y permite reducir el tiempo de desarrollo de las pruebas o test en más del 40%.

El trabajo realizado en la tesis ha tenido impacto no solo en publicaciones científicas, sino también en los proyectos del grupo de investigación. Estos aspectos serán estudiados en los próximos apartados.

7.1. Publicaciones

Las actividades de investigación realizadas en la presente tesis han servido de base a 22 aportaciones a la comunidad científica, en forma de:

- 3 Publicaciones en Revistas indexadas en *Journal Citation Reports (JCR).*
- 1 Patente concedida con examen previo.
- 10 Artículos presentados en conferencias internacionales indexadas.
- 3 Capítulos de libros.
- 5 aportaciones a conferencias internacionales sin publicación.

A continuación, se presentan las diferentes publicaciones que se han generado durante el desarrollo de la tesis, ordenadas por fecha y medio de publicación. En cada caso se indica la relación que tienen con la tesis. Las publicaciones realizadas durante el desarrollo del presente trabajo han sido:

1. **Revistas indexadas en *Journal Citation Reports (JCR):***
   a. P. Peñil, Á. Díaz, H. Posadas, P. Sánchez, " High-Level Design of Wireless Sensor Networks for Performance Optimization Under Security Hazards ", ACM Transactions on Sensor Networks (TOSN), 2017
      - Presenta el modelado en UML-MARTE y la simulación de redes de sensores inalámbricas cuando son atacadas. Es parte del trabajo presentado en el capítulo 3 de la tesis.

b. Á. Díaz, P. Sánchez, "Simulation of Attacks for Security in Wireless Sensor Network" Sensors 2016, 16(11), 1932.
   - Aúna el trabajo realizado en el entorno de análisis de prestaciones de redes de sensores inalámbricas, presentado en los capítulos 2 y 3.

c. Á. Díaz, J. González-Bayon, P. Sánchez, "Security Estimation in Wireless Sensor Network Simulator", Journal of Circuits, Systems and Computers 2016 25:07
   - Presenta la métrica de seguridad “SEM”, desarrollada en el capítulo 4 de esta tesis.

2. Patente:

a. Á. Díaz, P. Sánchez, "MÉTODO Y DISPOSITIVO PARA LA ACTUALIZACIÓN EFICIENTE DE DATOS EN DISPOSITIVOS ELECTRÓNICOS”, "Method and system for Embedded Systems Partial Firmware Update", ES2481343B2
   - Patenta el método de actualización de software presentado en la sección 5.1 de esta tesis.

3. Artículos en foros internacionales:

a. P. González-de-Aledo, Á. Díaz, P. Sánchez, R. Huuck, "Discovering and Validating Concurrency Specification from Test Executions," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suiza, 2016, pp. 5-8.
   - Introduce aspectos de la metodología de verificación presentada en el capítulo 6.

b. Á. Díaz, A. Nicolás, I. Ugarte, P. Sánchez, "Designing embedded HW/SW systems with OpenMP", FDL Forum on specification & Design Languages September 12-14, 2016 Bremen, Germany. 2016-09
   - Presenta una alternativa para aumentar la eficiencia de las aplicaciones mediante la generación automática desde OpenMP de código HW/SW. Este trabajo está relacionado con mejoras del entorno de co-simulación HW/SW descritos en el capítulo 2.

c. P. González-de-Aledo, L. Díaz, Á. Díaz, P. Sánchez, "Profiling and optimizations for embedded systems," 2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE), Lausanne, 2014, pp. 194-197
   - Se presentan resultados obtenidos con metodologías de análisis de prestaciones y verificaciones, descritas en los capítulos 2 y 6.

d. Cardona, L.A.,, S. de la Fe, B. Lorente, Villar,S., Guo, Y., Ferrer, C., Á. Díaz , J. González, P. Sánchez, J. Sancho, J. Rico, B. Cendón "Security of Low Power Wireless Sensor Meshed Network", 1st Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2013), Pope's Palace, Avignon, Francia, Contributed talk, 30/05/2013-31/05/2013
   - Este trabajo analiza los aspectos de seguridad que se deben tener en cuenta en las redes de sensores inalámbricas.

e. Á. Díaz., J. González-Bayon, P. González de Aledo Marugán, P. Sánchez, "Virtual platform for power and security analysis of wireless sensor network", Proc. SPIE 8764, VLSI Circuits and Systems VI, 87640I (May 28, 2013); doi:10.1117/12.2019253
   - En este artículo se presenta una primera aproximación a la métrica de seguridad descrita en el capítulo 4. Dicho trabajo será posteriormente ampliado y mejorado en una publicación en una revista indexada en JCR.

f. Á. Díaz, P. Sánchez, J. Sancho and J. Rico, "Wireless sensor network simulation for security and performance analysis," 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2013, pp. 432-435.
   - En este artículo se presenta una primera versión de la simulación de los ataques en un entorno de análisis de prestaciones (Capítulo 3). Únicamente incluye un pequeño número de ataques, que posteriormente fue mejorado y extendido.

g. Á. Díaz, P. Peñil, P. Sánchez, J. Sancho and J. Rico, "Modeling and simulation of secure wireless sensor network," Proceeding of the 2012 Forum on Specification and Design Languages, Vienna, 2012, pp. 185-192.
   - Este trabajo presenta una primera introducción al modelado de las redes y los ataques con lenguajes de alto nivel, como es UML-Marte. Cubre parte de lo presentado en el capítulo 3 de la tesis.

h. Á. Díaz, P. Sánchez, "Simulation of attacks in Wireless Sensor Network", XXVII Conference on Design of Circuits and Integrated Systems, DCIS'12. 2012-11
   - Se presenta una primera introducción a la simulación de seguridad en redes, presentado en el capítulo 3.

i. Á. Díaz, R. Diego and P. Sánchez, "Virtual Platform for Wireless Sensor Networks," 2012 15th Euromicro Conference on Digital System Design, Izmir, 2012, pp. 858-865. doi: 10.1109/DSD.2012.137
   - En este artículo se presenta el simulador de redes sobre el que se construye todo el entorno de análisis de prestaciones para redes de sensores inalámbricas. Principalmente presenta los trabajos que han servido de base al capítulo 2.

j. D. Calvo, J. Pérez, P. González, R. Diego, Á. Díaz, P. Sánchez "Design, modeling and development of an efficient communication infrastructure for networking applications" XXVI Conference on Design of Circuits and Integrated Systems, DCIS'11. 2011-11
   - Presenta una primera introducción a la comunicación eficiente en redes. Este trabajo sirvió de base al modelado de red del capítulo 2.

4. Capítulos de libros:

a. Á. Díaz, J. P. González, P. Sánchez, "Wireless Sensor Networks: Virtual Platform for Performance Analysis and Attack Simulation", Trusted Computing for Embedded Systems Springer International Publishing, pp. 247-269, 2015.

- Este capítulo de libro muestra el trabajo realizado en los capítulos 2 y 3 de la tesis y que permiten la simulación del comportamiento y análisis del impacto de ataques a redes de sensores inalámbricas.

b. J. Rico, J. Sancho, Á. Díaz, J. P. González, P. Sánchez, B. L. Sánchez Alvarez, L. A. C. Cardona, C. F. Ramis, "Low power wireless sensor networks: Secure applications and remote distribution of FW updates with key management on WSN", Trusted Computing for Embedded Systems Springer International Publishing, pp. 71-111, 2015.

- Presenta las técnicas desarrolladas para mejorar la eficiencia y la seguridad en redes de sensores inalámbricas. Este trabajo ha sido descrito en el capítulo 5 de la tesis.

a. H. Posadas, Á. Díaz and E. Villar (2012). SW Annotation Techniques and RTOS Modelling for Native Simulation of Heterogeneous Embedded Systems, Embedded Systems - Theory and Design Methodology, Dr. Kiyofumi Tanaka (Ed.), InTech.

- Este artículo introduce el modelado del RTOS en el simulador utilizado en el entorno de análisis de prestaciones. Contiene una descripción de alto nivel de la simulación de la API win32, mencionada en la sección 2.3.3 de esta tesis.

5. Otras aportaciones a conferencias internacionales sin publicación:

a. Á. Díaz, P. Sánchez “Attack-Aware firmware and power consumption demostration”. European Nanoelectronic Forum, (ENF2013) Barcelona 2013

- En este evento se presentó una demostración del firmware que evita ataques, desarrollado en la sección 3.8. Además, se

comparó el consumo de energía de los algoritmos de encriptación Hardware frente a los implementados en Software.

b. Á. Díaz, P. Sánchez et all, “TOISE Project Poster”, European Nanoelectronic Forum, (ENF2013), Barcelona
   - Póster del proyecto TOISE que será comentado en el próximo apartado. Además, se presentaron las técnicas desarrolladas y el simulador de redes de sensores inalámbricas.

c. Á. Díaz, P. Sánchez et all, “TOISE Project Poster”. University Booth DATE, 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden 2012
   - Poster en el University Booth sobre el desarrollo del proyecto TOISE, con presentación de las técnicas desarrolladas y el simulador de redes de sensores inalámbricas.

d. Á. Díaz, P. Sánchez et all, “TOISE Project Poster”, European Nanoelectronic Forum, (ENF2012) Munich 2012.
   - Poster sobre el desarrollo del proyecto TOISE, con presentación de las técnicas desarrolladas y el simulador de redes de sensores inalámbricas.

e. Á. Díaz, P. Sánchez et all, “TOISE Project Poster”, European Nanoelectronic Forum, (ENF2011) Dublin 2011
   - Poster sobre el desarrollo del proyecto TOISE, con presentación de las técnicas desarrolladas y el simulador de redes de sensores inalámbricas.

7.2. Proyectos de investigación

Los comienzos de este trabajo se enmarcan dentro de las actividades del proyecto del Plan Nacional DREAMS *(Dynamically Reconfigurable Embedded*

Platforms for Networked Context-Aware Multimedia Systems, TEC2011-28666-C04-02) y continúan en el proyecto REBECCA (Sistemas Electrónicos Empotrados Confiables para Control de Ciudades bajo Situaciones Atípicas, TEC2014-58036-C4-3-R). Dichos trabajos fueron la base de la participación del grupo de investigación en el Proyecto Europeo ENIAC TOISE (*Trusted Computing for European Embedded Systems*) financiado por el Ministerio de Ciencia e Innovación (EUI2010-04255). TOISE se centró en el estudio y desarrollo de soluciones seguras para aplicaciones críticas. Aspectos relacionados con el desarrollo del simulador fueron también utilizados en el proyecto europeo ARTEMIS 100029 SCALOPES (*SCalable LOw Power Embedded platformS*), financiado por el Ministerio de Industria, Energía y Turismo (ART-010000-2009-9) y ARTEMIS JU (CE). El trabajo de TOISE se continuó desarrollando dentro del proyecto europeo ARTEMIS 100371 CRAFTERS (*Constraint and application driven framework for tailorin embedded real-time systems*) financiado por el Ministerio de Industria, Energía y Turismo (ART-010000-2012-5) y ARTEMIS JU (CE). CRAFTERS tenía como objetivo garantizar el funcionamiento seguro y confiable del sistema, reduciendo el consumo sin aumentar el tiempo de ejecución. Por último, la experiencia adquirida en la tesis ha sido utilizada en el proyecto ARTEMIS EMC2 (*Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments*) para facilitar la certificación de la seguridad de sistemas embebidos. En este proyecto, el grupo de investigación ha participado como entidad subcontratada por un socio del proyecto.

Además de potenciar la participación en proyectos nacionales y europeos, esta tesis ha permitido abrir una nueva línea de investigación en el grupo, dedicada al diseño y análisis de la seguridad de sistemas embebidos.

Referencias

[1] **Informe de amenazas CCN-CERT IA-09/16 Ciberamenazas 2015 y Tendencias 2016**.

[2] **Informe de amenazas CCN-CERT IA-09/15 Ciberamenazas 2014 y Tendencias 2015**

[3] **Informe de amenazas CCN-CERT IA-03/14 Ciberamenazas 2013 y Tendencias 2014**

[4] **ENISA Threat Landscape 2015**
https://www.enisa.europa.eu/publications/etl2015
Accedido en abril de 2017.

[5] N. Mukherjee, S. Neogy, S. Roy.
Building Wireless Sensor Networks: Theoretical and Practical Perspectives
ISBN 9781482230062, CRC/Chapman & Hall.Taylor & Francis. November 24, 2015.

[6] T. Kavitha, D. Sridharan.
Security vulnerabilities in wireless sensor networks: A survey
Journal of Information Assurance and Security, 5 (1) (2010), pp. 31–44.

[7] D.P. Harrop, R. Das.
Wireless sensor networks (wsn) 2012–2022: forecasts, technologies, players,
IDTechEx, 2014.

[8] **Global Wireless Sensor Network Market, by Application (Transportation, Entertainment, Healthcare, Oil & gas, Food & beverage), by Sensors (Pressure, Temperature, Humidity, Flow), by Technologies (Bluetooth, Wi-Fi, WirelessHART)**
Forecast 2022.

[9] D.P. Harrop, R. Das.
Industrial wireless sensor networks (iwsn) market—global forecast & analysis (2012–2017): by technology, components, applications, geography,
MarketsandMarkets, 2014.

[10] **Industrial wireless sensor networks market worth $944.92 million by 2020**.
MarketsandMarkets
http://www.marketsandmarkets.com/PressReleases/wireless-sensor-network.asp
Accedido en abril de 2017.

[11] P. Harrop, R. Das.
Wireless sensor networks (wsn) 2014–2024: forecasts, technologies, players,
IDTechEx 2014

[12] G. Kumar , P. K. Bhatia.
Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies
2014 Fourth International Conference on Advanced Computing & Communication Technologies, 2014, pp. 189–196.

[13] M. Mekni, B. Moulin.
A survey on sensor webs simulation tools
Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications, (sensorcomm 2008), Cap Esterel, 2008, pp. 574-579.

[14] A. Stetsko, M. Stehlik and V. Matyas.
Calibrating and Comparing Simulators for Wireless Sensor Networks
2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, MASSIEEE (2011) Valencia, 2011, pp. 733-738.

[15] D. Curren.
A survey of simulation in sensor networks
University of Binghamton project report for subject CS580.

[16] P. Chhimwal, D. S. Rai, D. Rawat.
Comparison between Different Wireless Sensor Simulation Tools
IOSR Journal of Electronics and Communication Engineering (IOSR-JECE).Volume 5, Issue 2 (Mar. - Apr. 2013), PP 54-60.

[17] A. K. Pathan, M. M. Monowar, S. Khan.
Simulation Technologies in Networking and Communications: Selecting the Best Tool for the Test
ISBN 9781482225495, CRC Press, Taylor & Francis Group, USA, November 2014.

[18] N. I. Sarkar, S. A. Halim.
A Review of Simulation of Telecommunication Networks: Simulators, Classification, Comparison, Methodologies, and Recommendations
Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT), March Edition, 2011.

[19] H. Sundani, H. Li, V. K. Devabhaktuni, M. Alam, P. Bhattacharya.
Wireless Sensor Network Simulators A Survey and Comparisons
International Journal of Computer Networks (IJCN), Volume2: Issue 5, 2010, pp 25.

[20] **Contiki-os**
http://www.contiki-os.org
Accedido en abril de 2017.

[21] A. Boulis.
Castalia: Revealing pitfalls in designing distributed algorithms in WSN
Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07), pp. 407-408.

[22] T. Issariyakul, E. Hossain.
Introduction to Network Simulator Ns2
ISBN: 978-1-4614-1406-3, Springer Publishing Company, 2008.

[23] G. F. Riley, T. R. Henderson.
The NS-3 network simulator
ISBN: 978-3-642-12330-6, Modeling and Tools for Network Simulation. Springer Publishing Company, 2010, pp 15-34.

[24] **OMNeT++**
http://www.omnetpp.org
Accedido en abril de 2017.

[25] X. Zeng, R. Bagrodia, M. Gerla
GloMoSim: a library for parallel simulation of large-scale wireless networks
Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on, Banff, Alta., 1998, pp. 154-161.

[26] P. Levis, N. Lee.
TOSSIM: A Simulator for TinyOS Networks
Computer Science Division, University of California Berkeley, California, September 2003.

[27] B. L Titzer, J. Palsberg, D. K. Lee.
Avrora: Scalable sensor network simulation with precise timing
IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005, pp. 477-482.

[28] R. Bagrodia, R. Meyer,M. Takai, Y. Chen, X. Zeng, J. Martin, H. Y. Song
PARSEC: A Parallel Simulation Environment for Complex Systems
IEEE Computer, vol. 31, no. 10, pp. 77-85, Oct 1998.

[29] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler
TinyOS: An operating system for wireless sensor networks.
Ambient Intelligence. Springer-Verlag, 2004.

[30] L. Peng, J. zeng, H. Yuan, H. Li
WSM: Introduction, Design and Case Study. Wireless Communications
2007 International Conference on Wireless Communications, Networking and Mobile Computing (WiCom 2007), Shanghai, 2007, pp. 2580-2583.

[31] L. Berardinelli, V. Cortellessa, S. Pace
Modeling and Analyzing Performance of Software for Wireless Sensor Networks.
Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications (SESENA '11). ACM, New York, NY, USA, 13-18.

[32] A. Sobeih, J.C. Hou, L. C. Kung, N. Li, H. Zhang, W.-P. Chen, H.-Y. Tyan, H. Lim
J-Sim: A Simulation and Emulation Environment for Wireless Sensor Networks
IEEE Wireless Communications, vol. 13, no. 4, Aug. 2006, pp. 104-119.

[33] S. Dhurandher, S. Misra, M. Obaidat, S. Khairwal
UWSim: A simulator for underwater sensor networks
Simulation, vol. 84, no. 7, 2008, pp. 327-338.

[34] S. P. Fekete, A. Kroller, S. Fischer, D. Pfisterer
Shawn: The fast, highly customizable sensor network simulator
Proceedings of the Fourth International Conference on Networked Sensing Systems (INSS 2007), Braunschweig, 2007, pp. 299-299.

[35] G. Simon, P. Volgyesi, M. Maroti, A. Ledeczi
Simulation-based optimization of communication protocols for large-scale wireless sensor networks
2003 IEEE Aerospace Conference Proceedings (Cat. No.03TH8652), 2003, pp. 3_1339-3_1346.

[36] H. Adam, W. Elmenreich, C. Bettstetter, S. M. Senouci
CoRe-MAC: A MAC-protocol for cooperative relaying in wireless networks
Proceedings of the 2009 IEEE Global Communication Conference (Globecom), Honolulu, Hawaii, 2009.

[37] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, M. Karir
ATEMU: A fine-grained sensor network simulator
2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004, 2004, pp. 145-152.

[38] M. Gligor, N. Fournel, F. Pétrot.
Using binary translation in event driven simulation for fast and flexible MPSoC simulation
7th IEEE/ACM International Conference on Hardware/Software-Co-Design and System Synthesis, CODES+ISSS 2009, pp. 71-80.

[39] P. Botella, P. Sánchez, H. Posadas.
Automatic Generation of SystemC SMP Models for HW/SW Co-Simulation
Proceedings of the XXV Conf. on Design of Circuits and Integrated Systems, DCIS'10, 2010.

[40] P. Gerin, M. Hamayun, F. Petrot.
Native MPSoC co-simulation environment for software performance estimation
Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis (CODES+ISSS '09). ACM, New York, NY, USA, 403-412.

[41] H. Posadas, J. Castillo, D. Quijano, V. Fernandez, E. Villar, M. Martinez
SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems
L. Gomes, & J. Fernandes (Eds.), Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation, pp. 219-243.

[42] C. Pronk.
Verifying FreeRTOS; a feasibility study
Delft University of Technology, Software Engineering Research Group, Tech. Rep. TUD-SERG-2010-042, 2010.

[43] Adi Mallikarjuna V. Reddy, A.V.U. Phani Kumar, D. Janakiram, G. Ashok Kumar.
Wireless sensor network operating systems: a survey
International Journal of Sensor Networks, 2009 Vol. 5 No. 4, 236-255.

[44] H. Posadas, Á. Díaz, E. Villar
SW Annotation Techniques and RTOS Modeling for Native Simulation of Heterogeneous Embedded Systems
Kiyofumi Tanaka: "Embedded Systems - Theory and Design Methodology", InTech, Croatia. 2012.

[45] **Android, FreeRTOS top EE Times' 2013 embedded survey**
http://www.eetimes.com/document.asp?doc_id=1263083
Accedido en abril de 2017.

[46] **Wine**
http://www.winehq.org
Accedido en abril de 2017.

[47] **FreeRTOS**
http://www.freertos.org
Accedido en abril de 2017.

[48] R. P. Torres, L. Valle, M. Domingo, S. Loredo, M. C. Diez.
CINDOOR: an engineering tool for planning and design of wireless systems in enclosed spaces
IEEE Antennas and Propagation Magazine, vol. 41, no. 4, pp. 11-22, Aug. 1999.

[49] **XBee**
https://www.digi.com/products/xbee-rf-solutions/embedded-rf-modules-modems/xbee-802-15-4
Accedido en abril de 2017.

[50] **ARM Cortex-M3**
http://www.arm.com/products/processors/cortex-m/cortex-m3.php
Accedido en abril de 2017.

[51] G, Padmavathi, D. Shanmugapriya.
A survey of attacks, security mechanisms and challenges in wireless sensor networks
International Journal of Computer Science and Information Security, 2009, 4, 117–125.

[52] Y. Wang, G. Attebury, B. Ramamurthy.
A survey of security issues in wireless sensor networks
IEEE Communications Surveys & Tutorials, vol. 8, no. 2, pp. 2-23, Second Quarter 2006.

[53] J. P. Walters, Z. Liang; W. Shi; V. Chaudhary.
Wireless Sensor Network Security: A Survey
Security in Distributed, Grid and Pervasive Computing; Taylor & Francis: Oxfordshire, UK, 2006.

[54] D. W. Carman, P. S. Krus, B. J. Matt.
Constraints and approaches for distributed sensor network security
Technical Report 00-010, NAI Labs, Network Associates, Inc., Glenwood, MD, 2000.

[55] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, D. E. Culler.
Spins: security protocols for sensor networks
Wireless Networking, 8(5):521–534, 2002.

[56] M. Y. Malik.
An outline of security in wireless sensor networks: threats, countermeasures and implementations.
Wireless Sensor Networks and Energy Efficiency: Protocols, Routing and Management, 2011.

[57] J. Shukla, B. Kumari.
Security threats and defense approaches in wireless sensor networks: An overview
International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 8, August 2013, 165–175.

[58] H. L. Nguyen, U. T. Nguyen.
A study of different types of attacks on multicast in mobile ad hoc networks
Ad Hoc Networks, Volume 6, Issue 1, January 2008, Pages 32-46.

[59] S. Han, E. Chang, L. Gao, T. Dillon.
Taxonomy of Attacks on Wireless Sensor Networks
EC2ND 2005; Springer: London, UK, 2006; pp. 97–105.

[60] T. G. Lupu.
Main types of attacks in wireless sensor networks
Proceedings of the 9th WSEAS International Conference on Signal, Speech and Image Processing, and 9th WSEAS International Conference on Multimedia, Internet & Video Technologies; World Scientific and Engineering Academy and Society Stevens Point, WI, USA, 2009; pp. 180–185.

[61] P. Mohanty, S. A. Panigrahi, N. Sarma, S. S. Satapathy.
Security Issues in Wireless Sensor Network Data Gathering Protocols: A Survey
Journal of Theoretical and Applied Information Technology, 2010, pp. 14 – 27.

[62] C. Karlof, D. Wagner.
Secure routing in wireless sensor networks: attacks and countermeasures.
Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003, pp. 113-127.

[63] E. Huang, C. H. Scott, E. MacCallum, E. David, D. Z. Du.
Attacks and Countermeasures in Sensor Networks: A Survey
Springer: New York, NY, USA, 2010.

[64] S. Mohammadi, H. Jadidoleslamy.
A Comparison of Link Layer Attacks on WSN
International journal on applications of graph theory in wireless ad hoc networks and sensor networks. Vol. 3, No. 1, March 2011.

[65] Rajkumar; B. A. Vani, Rajaraman, H. G.; Chandrakanth.
Security Attacks and its Countermeasures in Wireless Sensor Networks.
International Journal of engineering Research and Applications. 2014, 4, 4–15.

[66] A. Díaz; J. González, P. Sánchez.
Security estimation in wireless sensor network simulator
Journal of Circuits, Systems and Computers Volume 25, Issue 07, July 2016.

[67] G. Dini, M. Tiloca.
On Simulative Analysis of Attack Impact in Wireless Sensor Networks
Proceedings of the 18th IEEE International Conference on Emerging Technology & Factory Automation (ETFA 2013), Cagliari, Italy, 10–13 September 2013.

[68] G. Dini, M. Tiloca.
ASF: An Attack Simulation Framework for wireless sensor networks
Proceedings of the 8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2012), Barcelona, Spain, 8–10 October 2012.

[69] L. Sánchez-Casado, R. A. Rodríguez-Gómez, R Magán-Carrión, G. Maciá-Fernández.
NETA: Evaluating the Effects of NETwork Attacks. MANETs as a Case Study
Advances in Security of Information and Communication Networks. Communications in Computer and Information Science, vol 381. Springer, Berlin, Heidelberg.

[70] T. Bonaci, L. Bushnell, R. Poovendran.
Node capture attacks in wireless sensor networks: A system theoretic approach
Proceedings of the 49th IEEE Conference on Decision and Control (CDC 2010), Atlanta, GA, USA, 15–17 December 2010.

[71] Y. L. Huang, A. A. Cardenas, S. Amin, Z. S. Lin, H. Y. Tsai, S. Sastry.
Understanding the physical and economic consequences of attacks on control systems
International Journal of Critical Infrastructure Protection. 2009, 2, 73–83.

[72] Y. Xu, G. Chen, J. Ford, F. Makedon.
Detecting Wormhole Attacks in Wireless Sensor Networks.
Critical Infrastructure Protection; Springer: New York, NY, USA, 2007.

[73] S. Kaplantzis, A. Shilton, N. Mani, Y. A. Sekercioglu.
Detecting Selective Forwarding Attacks in Wireless Sensor Networks using Support Vector Machines
Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, Melbourne, Australia, 3–6 December 2007.

[74] Y. T. Wang, R. Bagrodia.
SenSec: A Scalable and Accurate Framework for Wireless Sensor Network Security Evaluation
Proceedings of the 31st International Conference on Distributed Computing Systems Workshops (ICDCSW 2011), Minneapolis, MN, USA, 20–24 June 2011.

[75] S. Hong, S. Lim.
Analysis of Attack Models via Unified Modeling Language in Wireless Sensor Networks: A Survey Study
Wireless Communications, Networking and Information Security (WCNIS). In Proceedings of the 2010 IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS), Beijing, China, 25–27 June 2010.

[76] K. Pelechrinis, M. Iliofotou, S. V. Krishnamurthy.
Denial of Service Attacks in Wireless Networks: The Case of Jammers
EEE Communications Surveys & Tutorials, vol. 13, no. 2, pp. 245-257, Second Quarter 2011.

[77] P. Reindl, K. Nygard, X. Du.
Defending Malicious Collision Attacks in Wireless Sensor Networks
2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Hong Kong, 2010, pp. 771-776.

[78] A.D. Wood, J.A. Stankovic.
A Taxonomy for Denial-of-Service Attacks in Wireless Sensor Networks
ISBN: 978-0-8493-1968-6 Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, CRC Press 2004.

[79] J. Deng, Richard Han, Shivakant Mishra.
Defending against path-based DoS attacks in Wireless Sensor Networks
Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks (SASN '05). ACM, New York, NY, USA, 89-96.

[80] M. Yasir.
An Outline of Security in Wireless Sensor Networks Threats, Countermeasures and Implementations
Wireless Sensor Networks and Energy Efficiency: Protocols, Routing and Management Book, 2012 pp. 507-527.

[81] S. Ramaswamy, H. Fu, M. Sreekantaradhya, J. Dixon, K. Nygard.
Prevention of Cooperative Black Hole Attack in Wireless Ad Hoc Networks
Int'l Conf. on Wireless Networks, 2003.

[82] A. Dubey, V. Jain, A. Kumar.
A Survey in Energy Drain Attacks and Their Countermeasures in Wireless Sensor Networks
International Journal of Engineering Research & Technology Vol. 3 - Issue 2 February, 2014.

[83] R. Singh, J. Singh, R. Singh.
Hello flood attack countermeasures in wireless sensor networks
International Journal of Computer Science and Mobile Applications, Vol.4 Issue. 5, May- 2016, pg. 1-9.

[84] M. Y. Abdullah G. W. Hua, N. Alsharabi.
Wireless sensor networks misdirection attacker challenges and solutions
2008 International Conference on Information and Automation, Changsha, 2008, pp. 369-373.

[85] A. Dubey, D. Meena, S. Gaur.
A Survey in Hello Flood Attack in Wireless Sensor Networks
International Journal of Engineering Research & Technology Vol. 3 - Issue 1 January, 2014.

[86] I. Krontiris, T. Giannetsos, T. Dimitriou.
Launching a sinkhole attack in wireless sensor networks; the intruder side
Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication. Washington, DC, USA: IEEE Computer Society, 2008, pp. 526–53.

[87] K. Jindal, S. Dalal, K. K. Sharma.
Analyzing Spoofing Attacks in Wireless Networks
2014 Fourth International Conference on Advanced Computing & Communication Technologies, Rohtak, 2014, pp. 398-402.

[88] J. Newsome, E. Shi, D. Song, A. Perrig.
The sybil attack in sensor networks: analysis & defenses
Proceedings of the 3rd international symposium on Information processing in sensor networks, April 26-27, 2004, Berkeley, California, USA.

[89] H. R. Shaukat, F. Hashim, A. Sali, M. F. Abdul Rasid.
Node Replication Attacks in Mobile Wireless Sensor Network: A Survey
International Journal of Distributed Sensor Networks Volume 2014, Article ID 402541, 15 pages.

[90] W. Z. Khan, M. Y. Aalsalem, M. N. Bin, M. Saad, Y. Xiang.
Detection and Mitigation of Node Replication Attacks in Wireless Sensor Networks: A Survey
International Journal of Distributed Sensor Networks Volume 2013, Article ID 149023, 22 pages.

[91] X. Chen, L. Meng, Y. Zhan.
Detecting and Defending against Replication Attacks in Wireless Sensor Networks
International Journal of Distributed Sensor Networks Volume 2013, Article ID 240230, 10 pages.

[92] T. Kavitha, D. Sridharan.
Security vulnerabilities in wireless sensor networks: A survey
Journal of Information Assurance and Security, 5:31-44.

[93] A. Becher, Z. Benenson, M. Dornseif.
Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks
Clark J.A., Paige R.F., Polack F.A.C., Brooke P.J. (eds) Security in Pervasive Computing. SPC 2006. Lecture Notes in Computer Science, vol 3934. Springer, Berlin, Heidelberg.

[94] P. Peñil, A. Diaz, H. Posadas, P. Sánchez
High-Level Design of Wireless Sensor Networks for Performance Optimization Under Security Hazards
ACM Transactions on Sensor Networks (TOSN), 2017

[95] K. J. Higgins.
New Technology Detects Cyberattacks By Power Consumption
http://www.eetimes.com/author.asp?section_id=36&doc_id=1325409
Accedido en abril de 2017.

[96] **Rzusbstick**
http://www.atmel.com/tools/rzusbstick.aspx
Accedido en abril de 2017.

[97] **Dc-power-analyzer-modular**
http://www.home.agilent.com/en/pd-1842303-pn-N6705B
Accedido en abril de 2017.

[98] O. Al-Jarrah, R. Saifan.
A novel algorithm for defending path-based denial of service attacks in sensor networks
Hindawi International Journal of Distributed Sensor Networks, 2010.

[99] J. Portilla, A. Otero, E. De La Torre, T. Riesgo, O. Stecklina, S. Peter, P. Langendörfer. **Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors**
International Journal of Distributed Sensor Networks 740823(12), October 2010.

[100] B. Wang.
Sensor Placement for Complete Information Coverage in Distributed Sensor Networks
Journal of Circuits, Systems and Computers, vol. 17, n 4, 2008. 627-636.

[101] H. O. Alanazi, B. B. Zaidan, A. A. Zaidan, H. A. Jalab, M. Shabbir, Y. Al-Nabhani .
New Comparative Study Between DES, 3DES and AES within Nine Factors
Journal of Computing, Volume 2, Issue 3, March 2010.

[102] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback.
Report on the development of the Advanced Encryption Standard (AES) .
Journal of Research of the National Institute of Standards and Technology, Volumen 106, Number 3, May-Jun 2001.

[103] L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, N. A. Heckert, J. F. Dray, S. Vo.
A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.
Technical Report. NIST Special Publication 800-22, National Institute of Standards and Technology, Gaithersburg, MD, United States.

[104] J. Soto.
Statistical Testing of Random Number Generators
Proceedings of the 22nd National Information Systems Security Conference, 1999.

[105] H. O. Alanazi, B. B. Zaidan, A. A. Zaidan, H. A. Jalab, M. Shabbir, Y. Al-Nabhani .
New Comparative Study Between DES, 3DES and AES within Nine Factors
Journal of Computing, Volume 2, Issue 3, March 2010.

[106] P. Prasithsangaree, P. Krishnamurthy.
Analysis of Energy consumption of RC4 and AES algorithms in wireless LANs
Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE, 2003, pp. 1445-1449 vol.3.

[107] W. Liu, R. Luo, H. Yang.
Cryptography Overhead Evaluation and Analysis for Wireless Sensor Networks
2009 WRI International Conference on Communications and Mobile Computing, Yunnan, 2009, pp. 496-501.

[108] B. Y. Ryabko, V.S. Stognienko, Y. I. Shokin.
A new test for randomness and it application to some cryptographic problems
Journal of Statistical Planning and Inference 123 (2004) 365-376.

[109] S. Oshima, T. Nakashima, T. Sueyoshi.
Comparison of Properties between Entropy and Chi-Square Based Anomaly Detection Method
2011 14th International Conference on Network-Based Information Systems, Tirana, 2011, pp. 221-228.

[110] **Announcing the Advanced Encryption Standard (AES),**
Federal Information Processing Standards Publication 197, 2001.

[111] R. Fjeldstad, W. Hamlen.
Application program maintenance study: Report to our correspondents
Tutorial on Software Maintenance. IEEE Computer Press Society, 1983.

[112] B. Lientz, E. Swanson.
Problems in application software maintenance
Communications of the ACM, 24(11): 763–769, 1981.

[113] G.Denys, F.Piessens, F.Matthijs.
A survey of customizability in operating systems research
ACM Computing Surveys, 34(4):450–468, 2002.

[114] Á. Díaz, P. Sánchez
MÉTODO Y DISPOSITIVO PARA LA ACTUALIZACIÓN EFICIENTE DE DATOS EN DISPOSITIVOS ELECTRÓNICOS, Method and system for Embedded Systems Partial Firmware Update
ES2481343B2

[115] J. W. Hui, D. Culler.
The dynamic behavior of a data dissemination protocol for network programming at scale
Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SENSYS\'04), 2004.

[116] T. Stathopoulos, J. Heidemann, D. Estrin.
A Remote Code Update Mechanism for Wireless Sensor Networks
Technical report, DTIC Document (2003).

[117] P. K. Dutta, J. W. Hui, D. C. Chu, D. E. Culler.
Securing the deluge Network programming system
Proceedings of the 5th international conference on Information processing in sensor networks (IPSN '06). ACM, New York, NY, USA, 326-333.

[118] J. Jeong, D. Culler.
Incremental network programming for wireless sensors
Proc. 1st IEEE Commun. Soc. Conf. Sensor and Ad Hoc Communications and Networks (SECON\04) 2004.

[119] N. Reijers, K. Langendoen.
Efficient code distribution in wireless sensor networks
Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications (WSNA '03). ACM, New York, NY, USA, 60-67.

[120] J. Koshy, R. Pandey.
Remote Incremental Linking for Energy-Efficient Reprogramming of Sensor Networks
Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005., 2005, pp. 354-365.

[121] J. Koshy, R. Pandey.
VM*: Synthesizing scalable runtime environments for sensor networks
Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SENSYS\'05), 2005.

[122] A. Dunkels , N. Finne , J. Eriksson, T. Voigt.
Run-time dynamic linking for reprogramming wireless sensor networks
Proc. 4th ACM Conf. Embedded Networked Sensor Systems (SENSYS\'06), 2006.

[123] P. Levis, D. Culler.
Mate: A virtual machine for tiny networked sensors
Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems, 2002, pp.85 -95.

[124] R. Balani, C. C. Han , R. K. Rengaswamy , I. Tsigkogiannis, M. Srivastava.
Multi-level software reconfiguration for sensor networks
Proceedings of the 6th ACM & IEEE International conference on Embedded software (EMSOFT '06). ACM, New York, NY, USA, 112-121.

[125] C. C. Han, R. Kumar, R. Shea, E. Kohler, M. Srivastava.
A Dynamic Operating System for Sensor Nodes
Proceedings of the 3rd international conference on Mobile systems, applications, and services (MobiSys '05). ACM, New York, NY, USA, 163-176.

[126] R. K. Panta, S. Bagchi, S. Midkiff.
Zephyr: Efficient incremental reprogramming of sensor nodes using function call indirections and difference computation
Proceedings of the 2009 conference on USENIX Annual technical conference (USENIX'09). USENIX Association, Berkeley, CA, USA, 32-32.

[127] **Eclipse**
https://eclipse.org/
Accedido en abril de 2017.

[128] **Xynergy Board**
http://embedded-computing.com/products/id/?186251887184183131#
Accedido en abril de 2017.

[129] B. Zhao; H. Zhang; Z. Li.
A trusted start-up based on embedded system
2009 Ninth IEEE International Conference on Computer and Information Technology, Xiamen, 2009, pp. 242-246.

[130] L. H. Adnan, Y. M. Yussoff, H. Hashim
Secure boot process for wireless sensor node,
2010 International Conference on Computer Applications and Industrial Electronics, Kuala Lumpur, 2010, pp. 646-649.

[131] L. H. Adnan, H. Hashim, Y. M. Yussoff, M. U. Kamaluddin.
Root of trust for trusted node based-on ARM11 platform
The 17th Asia Pacific Conference on Communications, Sabah, 2011, pp. 812-815.

[132] J. Toldinas, V. Stuikys, M. Banionis.
Energy Efficiency Comparison with Cipher Strength of AES and Rijndael Cryptographic Algorithms in Mobile Devices
Electronics & Electrical Engineering;2011, Issue 108, p11, February 2011.

[133] H. Rifa-Pous J. Herrera-Joancomartí.
Computational and Energy Costs of Cryptographic Algorithms on Handheld Devices
Journal Computational and Energy Costs of Cryptographic Algorithms on Handheld Devices. Future Internet 2011, 3, 31-48.

[134] M. H. Eldefrawy, M. K. Khan, K. Alghathbar.
A key agreement algorithm with rekeying for wireless sensor networks using public key cryptography
2010 International Conference on Anti-Counterfeiting, Security and Identification, Chengdu, 2010, pp. 1-6.

[135] **How secure is AES against brute force attacks?**
http://www.eetimes.com/document.asp?doc_id=1279619
Accedido en abril de 2017.

[136] **MSO9404A Mixed Signal Oscilloscope**
http://www.keysight.com/en/pdx-x201764-pn-MSO9404A
Accedido en abril de 2017.

[137] IEC 61508-3:2013
Seguridad funcional de los sistemas eléctricos/electrónicos/electrónicos programables relacionados con la seguridad. Parte 3: Requisitos del software
AENOR

[138] **GoogleTest**
https://code.google.com/p/googletest/
Accedido en abril de 2017.

[139] L. Diaz, E. Gonzalez, E. Villar, P. Sanchez.
VIPPE, parallel simulation and performance analysis of multi-core embedded systems on multi-core platforms
Design of Circuits and Integrated Circuits (DCIS), 2014 Nov. 2014.

[140] J. A. Whittaker, J. Arbon, J. Carollo .
How Google Tests Software
ISBN: 978-0321803023 Addison-Wesley Professional, 2012.

[141] N. Llopis.
Games from within: exploring the c++ unit testing framework jungle
http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle.
Accedido en abril de 2017.

[142] S. Siegl, K. S. Hielscher, R. German, C. Berger.
Formal specification and systematic model-driven testing of embedded automotive systems
2011 Design, Automation & Test in Europe, Grenoble, 2011, pp. 1-6.

[143] M. J. Karlesky, W. I. Bereza, C. B. Erickson.
Effective Test Driven Development for Embedded Software
2006 IEEE International Conference on Electro/Information Technology, East Lansing, MI, 2006, pp. 382-387.

[144] J. Boydens, P. Cordemans, E. Steegmans.
Test-Driven Development of Embedded Software
European Conference on the Use of Modern Information and Communication Technologies, 2010.

[145] E. Shamsoddin-Motlagh.
A Survey of Test Framework
International Journal of Computer Applications (0975 – 8887). Volume 105 – No. 4, November 2014.

[146] P.C. Jorgensen
Software Testing: A Craftsman's Approach
CRC Press (2013)

[147] M. Wahid, A. Almalaise.
JUnit Framework: An Interactive Approach for Basic Unit Testing Learning in Software Engineering
3rd International Congress on Engineering Education (ICEED). 159-164.

[148] B. Wang, C. Zhu, J. Sheng.
MDA-based automated generation method of test cases and supporting framework
2010 2nd International Conference on Computer Engineering and Technology, Chengdu, 2010, pp. V1-106-V1-109.

[149] F. M. Manrique.
Testbricks: software framework for IEEE standard 1641 test programs
IEEE Instrumentation & Measurement Magazine, vol. 16, no. 4, pp. 34-39, August 2013.

[150] Z. Liu, Q. Chen, X. Jiang.
A Maintainability Spreadsheet-Driven Regression Test Automation Framework
2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, NSW, 2013, pp. 1181-1184.

[151] Y. Wu, Z. Yu.
Study of Software Reliability Test Application Framework
2010 International Conference of Information Science and Management Engineering, Xi'an, 2010, pp. 162-165.

[152] D. Franke, C. Weise.
Providing a Software Quality Framework for Testing of Mobile Applications
Fourth IEEE International Conference on Software Testing, Verification and Validation. 431-434.

[153] D. Zhang, D. Liu, C. Csallner, D. Kung, Y. Lei.
A distributed framework for demand-driven software vulnerability detection
The Journal of Systems and Software 87, January 2014, 60-73.

[154] IEEE 1800.2-2017
IEEE Approved Draft Standard for Universal Verification Methodology Language Reference Manual
IEEE Computer Society

[155] K. Salah.
A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities
2014 9th International Design and Test Symposium (IDT), Algiers, 2014, pp. 94-99.

[156] **IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language**
IEEE STD 1800-2009 , vol., no., pp.1-1285, Dec. 11 2009.

[157] **Portable-stimulus**
http://accellera.org/activities/working-groups/portable-stimulus
Accedido en abril de 2017.

[158] **Accellera**
http://accellera.org
Accedido en abril de 2017.

[159] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, M. Olivieri.
MPARM: Exploring the Multi-Processor SoC Design Space with SystemC
J. VLSI Signal Process. Syst. 41, 2 (September 2005), 169-182.

[160] M. C. Chiang, T. C. Yeh, G.-F. Tseng.
A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, V.30, N.4, April 2011.

[161] S. Chakravarty, Z. Zhao, A. Gerstlauer.
Automated, retargetable back-annotation for host compiled performance and power modeling
2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), ACM, 2013.

[162] G. Meszaros
(2007) Xunit Test Patterns.
Refactoring Test Code, Pearson Education

[163] **SQLITE**
https://sqlite.org
Accedido en abril de 2017.

[164] **ODROID-XU4**
http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825
Accedido en abril de 2017.